SPEC
Systems & Processes Engineering Corporation
401 Camp Craft Road • Austin, Texas 78746-6558
Phone: 512/306-1100 • Fax: 512/306-1122

Final Report

32-Bit Emulator Chip for High Throughput Processing

CONTRACT NUMBER:
N00019-96-C-0036

CDRL A002

Prepared for:

NAWCADWAR Commander, NAWCAD Warminster
Attn: Code 4.5.5.1.2
Warminster, PA 18974-0591

Prepared by:

Systems & Processes Engineering Corporation (SPEC)
401 Camp Craft Road
Austin, Texas 78746-6558

June 10, 1996

Table of Contents

32-Bit Emulator Chip for High Throughput Processing

1.0 Introduction

System & Processes Engineering Corporation (SPEC) has developed a high throughput emulator chip which provides direct execution of existing AN/AYK-14 standard code on the commercially available PowerPC microprocessor. The SPEC emulator chip allows legacy and PowerPC code to be executed simultaneously, supporting an easy migration path to a full PowerPC code implementation. The SPEC emulator chip has better performance than a traditional software emulation or ROM based design. The SPEC emulator chip has lower fabrication and maintenance cost than using a multiple parallel processor approach. In summary, the SPEC emulator chip has the following features:

- Full AN/AYK-14 emulation
- Simultaneously integrates legacy and new native PowerPC code
- Requires no software modifications
- Direct replacement of the AN/AYK-14 processor card
- Ease in performance validation
- Low cost and maintenance

SPEC can develop drop in replacement circuit cards to meet specific user applications based on its ASIC design implementation. Emulator ASICs can either be implemented in CMOS or GaAs providing up to 500 MHz operation.

As shown below, AN/AYK-14 instructions are fetched from main memory and decoded to form PowerPC instruction sequences. Upon receiving the instruction, the instruction decoder generates the operand address and the commands to direct instruction execution. For a simple instruction such as AND, OR, etc., the instructions are completed in the logic unit. For a complex instruction such as multiply, floating point arithmetic, etc., the instruction decoder sends the command signal to the ROM containing PowerPC instruction sequences, controller, and cycle counter.

The ROM sends out the proper preprogrammed PowerPC instruction opcodes and the controller aligns the data, addresses, register number, and register content to the PowerPC interface register. The cycle counter directs the controller and ROM to send data and opcodes to the PowerPC interface register, indicates the end of the instruction sequence, and signals the instruction fetch to request a new instruction.

SPEC has analyzed the AN/AYK-14 instruction set and determined that most instructions can be readily implemented as sequences of PowerPC instructions. The SPEC emulator chip can directly decode AN/AYK-14 cascade addressing mode in 5 clock cycles, and convert AN/AYK-14 floating point format to the Power PC double precision floating point format in a single clock cycle. Since they can be easily implemented in hardware, specialized instructions requiring external signal inputs are directly executed by the emulator chip.

Since the emulator chip is designed to provide a bridge from AN/AYK-14 technology to mainstream COTS RISC technology, a bypass mode is provided for operation in native mode, allowing use of software compiled directly to the PowerPC instruction set.

During the phase I program, SPEC has studied two different emulator designs. The first one is software design or ROM based design. The second design, which is modified from the ROM based design to target the AN/AYK-14, is called pseudo emulator design.

SPEC has identified the infeasibility of using software or ROM based design to direct the PowerPC in emulating the AN/AYK-14 computer. This infeasibility is due to the uniqueness of the AN/AYK-14 addressing mode and floating point format. The ROM based design requires the PowerPC to spend 52 clock cycles to complete the AN/AYK-14 cascade addressing decode, and 19 clock cycles to convert the AN/AYK-14 floating point format to PowerPC format. The SPEC pseudo emulator chip can directly decode the AN/AYK-14 cascade addressing mode in 5 clock cycles, and convert the AN/AYK-14 floating point format to the PowerPC double precision floating point format in single clock cycle.

SPEC proposes a Phase II follow-on program that accomplishes the following:

1) Finalize AN/AYK-14 emulator and PowerPC design requirements

2) Perform emulator chip design optimization

3) Fabricate and test the emulator chip

4) Design, fabricate, and test an AN/AYK-14 emulator PowerPC circuit card assembly

5) Perform validation testing

1.1 Problems associated with traditional software or ROM based approach

This section discusses the limitations of a traditional ROM based approach, which are overcome in SPEC's Pseudo Emulator Design described in Section 2.0.

The block diagram of the ROM based design is shown in Figure 1. The main components of the ROM based design are: the AN/AYK-14 instruction fetch and instruction decoder, ROM containing PowerPC instruction sequences, controller, and address lookup table. When receiving a new instruction, the instruction decoder sends the signal to the ROM and controller. The controller directs the ROM sending out the instruction sequence which decodes the addressing mode, and then executes the instructions. The address lookup table stores the AN/AYK-14 instruction address and the equivalent PowerPC address. These addresses are required when emulating a jump or branch instruction.

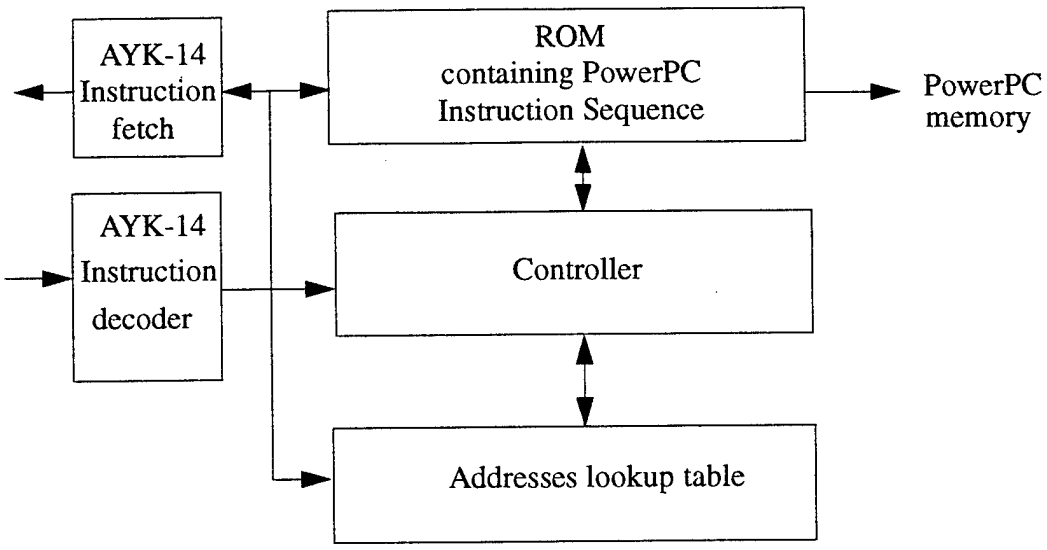

Figure 1 - The block diagram of the ROM based design.

1.1.1 Addressing mode.

PowerPC is a reduced instruction set computing (RISC) microprocessor. The PowerPC has a small instruction set and has a more simple addressing mode than the AN/AYK-14. There are three addressing modes in the PowerPC. Register indirect with immediate index mode (Y= (Ra) + d), register indirect with index mode (Y = (Ra) + (Rb)), and register indirect mode (Y = (Ra)). Whereas the AN/AYK-14 has direct, indirect, cascaded indirect, and cascaded indexed addressing modes. The direct and indirect addressing modes are easy to emulate. However, it requires a sequence of 52 PowerPC instructions to emulate the cascaded indirect and indexed addressing modes. The AN/AYK-14 cascaded indirect and indexed addressing modes are summarized below.

m = 0 operand at Y = y

m = 1-7,9,B,D,F operand at Y = y + (Rm)

m = 8, if SR2[9,8]= 0x operand at Y= (Rm)
if SR2[9,8]= 10, Let IW1 = (y), IW2 = (y+1)
if IW1[14,13,12] = 000 Then operand Y = IW2
if IW1[14,13,12] = 001 Then operand Y = IW2 + (Rx)
if IW1[14,13,12] = 010 Then operand Y = IW2 + (Rm)
if IW1[14,13,12] = 011 Then operand Y = IW2 + (Rm +1)
if IW1[14,13,12] = 100 Then operand Y = (IW2)
if IW1[14,13,12] = 101 Then operand Y = (IW2 + (Rx))
if IW1[14,13,12] = 110 Then operand Y = (IW2 + (Rm))

if IW1[14,13,12] = 111 Then operand Y = (IW2 + (Rm +1))
if SR2[9,8]= 11, Let IW1 = (y + (Rm)), IW2 = (y+1 + (Rm))
if IW1[14,13,12] = 000 Then operand Y = IW2
if IW1[14,13,12] = 001 Then operand Y = IW2 + (Rx)
if IW1[14,13,12] = 010 Then operand Y = IW2 + (Rm)
if IW1[14,13,12] = 011 Then operand Y = IW2 + (Rm +1)
if IW1[14,13,12] = 100 Then operand Y = (IW2)
if IW1[14,13,12] = 101 Then operand Y = (IW2 + (Rx))
if IW1[14,13,12] = 110 Then operand Y = (IW2 + (Rm))
if IW1[14,13,12] = 111 Then operand Y = (IW2 + (Rm +1))

The above decoded processes are carried out when m=A,C, or E, except the bit positions in the status register 2 (SR2) will change from [11,10], [13,12], or [15,14] respectively.

The 52 PowerPC instructions in the sequence below are used to emulate the cascade indirect, and indexed addressing modes.

```
1  LHA R24,d(r0)
2  LHA R29,d+1(r0);  Y=d+1
3  RLWINM R31, R24,SH=0,MB=28,ME=31
4  BC  (true goto 53)
5  COMPI  crfD,0,R31, simm=8
6  BC  (true goto 20)
7  COMPI  crfD,0,R31, simm=A
8  BC  (true goto 18)
9  COMPI  crfD,0,R31, simm=C
10 BC  (true  goto 16)
11 COMPI  crfD,0,R31, simm=E
12 BC  (true  goto  14)
13 B  goto 50
14 RLWINM R30, Rs2,SH=0,MB=22,ME=24
15 B  goto 21
16 RLWINM R30, Rs2,SH=0,MB=20,ME=22
17 B  goto 21
18 RLWINM R30, Rs2,SH=0,MB=18,ME=20
19 B  goto 21
20 RLWINM R30, Rs2,SH=0,MB=162,ME=18
21 COMPI  crfD,0,R30, simm=0 n[16]
22 BC end
23 COMPI  crfD,0,R30, simm=10 n[16,17]
24 BC  (true goto 27)
25 COMPI  crfD,0,R30, simm=11 n[16,17]
26 ADDO R29, R29, Rm
27 LHA R28,d(r29);  d=0
28 LHA R26,d(r29);  d=1
29 RLWINM R27, R28,SH=0,MB=18,ME=19
30 COMPI  crfD,0,R27, simm=0000
```

```
31  BC  (true goto 42)
32  COMPI  crfD,0,R27, simm=1000
33  BC  (true goto 41)
34  COMPI  crfD,0,R27, simm=2000
35  BC  (true goto 39)
36  COMPI  crfD,0,R27, simm=3000
37  ADDO R26, R26, Rm+1
38  B  goto 42
39  ADDO R26, R26, Rm
40  B  goto 42
41  ADDO R26, R26, Rx
42  RLWINM R27, R28,SH=0,MB=17,ME=17
43  COMPI  crfD,0,R27, simm=4000
44  BC  (false goto  48)
45  LHA R25, d(r26);   d=0
46  LHA R24, d(r25);   d=0
47  B  goto END
48  LHA R24, d(r26);   d=0
49  B  goto END
50  ADDIC R26, Rm, Y
51  LHA R24,d(r26);  d=0
52  B  goto END
53  LHA R24,d(r0);  d=Y
54  END
```

1.1.2 Floating point data format:

The AN/AYK-14 has a different floating point data format than the PowerPC. The AN/AYK-14 uses 32-bits (2 words) to represent floating point data, which can be broken down to 1 sign bit, 7 exponent bits, and 24 mantissa bits. The exponent is a hexadecimal (base16) number, and is biased by 64. The exponent number above 64 is a positive exponent and below 64 is a negative exponent. The normalization in the AN/AYK-14 is defined as the most significant hexadecimal digit that is non-zero. Whereas the PowerPC is a binary (base 2) number. The normalization in the PowerPC defines the most significant digit as one. The PowerPC single precision floating point has 23 mantissa bits which does not meet the resolution requirement of the AN/AYK-14. The PowerPC double precision floating point format which has a 52 bits mantissa is required to emulate the AN/AYK-14 floating point format. The exponent of the PowerPC double precision floating point has 11-bits and is biased by 1023.

A conversion process is required to translate AN/AYK-14 floating point data into the PowerPC double precision data format. The conversion algorithm is summarized bellow.

1. Subtract 64 from the exponent.

2. Multiply the exponent result by 4.

3. Add 1023 to the product.

4. Nomalize the mantissa.

5. Realign the exponent and mantissa into 64-bit format.

The conversion process requires 19 PowerPC instruction cycles to load the AN/AYK-14 floating point data into PowerPC double precision format. Similarly, it also requires 19 PowerPC instruction cycles to convert the PowerPC double precision data to AN/AYK-14 floating point data. The 19 PowerPC instructions in the sequence that converse the data format is listed below:

```
1  LHA Rm,d(r0)
2  RLWINM R31, Rm,SH=0,MB=1,ME=6
3  RLWINM R31, R31,SH=7,MB=25,ME=31
4  SUBFIC  R31, R31, SIMM=64
5  NEG R31, R31
6  RLWINM R31, R31,SH=3,MB=22,ME=28
7  ADDI R31, R31, SIMM=1028
8  RLWINM R30, Rm,SH=0,MB=0,ME=0
9  RLWINM R29, Rm,SH=12,MB=0,ME=31
10 CNTLZW  R28, R29
11 SUBF  R31, R28, R31
12 RLWNM R29, R29, R28, MB=0, ME=31
13 RLWINM R27, R29,SH=20,MB=0,ME=11
14 RLWINM R29, R29,SH=20,MB=12,ME=31
15 OR  R31, R30, R31
16 OR  R31, R31, R29
17 STW  R31, d(r0)
18 STW  R27, d+1(r0)
19 LFD  frD, d(r0)
```

1.1.3 Conditional Code

The load instructions in AN/AYK-14 require resetting the carry and overflow bits in the conditional register, whereas the load instructions in the PowerPC do not update the conditional code. A 'move to condition register from XER instruction is required to clear the carry and overflow bits in the XER register. In addition, the AN/AYK-14 has 16-bit data and the PowerPC has a 32-bit ALU. To have a correct carry out and overflow results, the 16-bit operands have to be in the 16 high order bits of the ALU, which requires two left shift instructions. A right shift is required to move the ALU result to its correct position.

1.1.4 Lookup Table.

Because of the expansion of the PowerPC instruction codes when emulating the AN/AYK-14 instruction codes, the conventional ROM based emulator design requires a lookup table to track the corresponding instruction address between the AN/AYK-14 and the PowerPC. This lookup table will generate the correct instruction address for the PowerPC when executing a jump, branch, or interrupt instruction. The size of this lookup table depends on the size of the application program.

A direct use of the PowerPC to emulate the AN/AYK-14 has a severe performance degradation and a large memory overhead for the lookup table. A hardware or pseudo emulator approach is required to speed up the emulation of AN/AYK-14 standard code and to reduce the memory overhead.

1.2 Pseudo Emulator Design.

SPEC has designed an pseudo emulator chip which provides a faster addressing mode decode and floating point data conversion. This emulator chip does not require a lookup table. The system overview of the chip is shown in Figure 2. The emulator chip design provides a bridge between the AN/AYK-14 processor and the PowerPC microprocessor. The emulator chip can be turned off when the PowerPC executes PowerPC programs, and turned on to direct the PowerPC to execute AN/AYK-14 legacy programs. The switching circuits which connect the PowerPC with the emulator chip can be embedded inside the emulator chip or designed at board level.

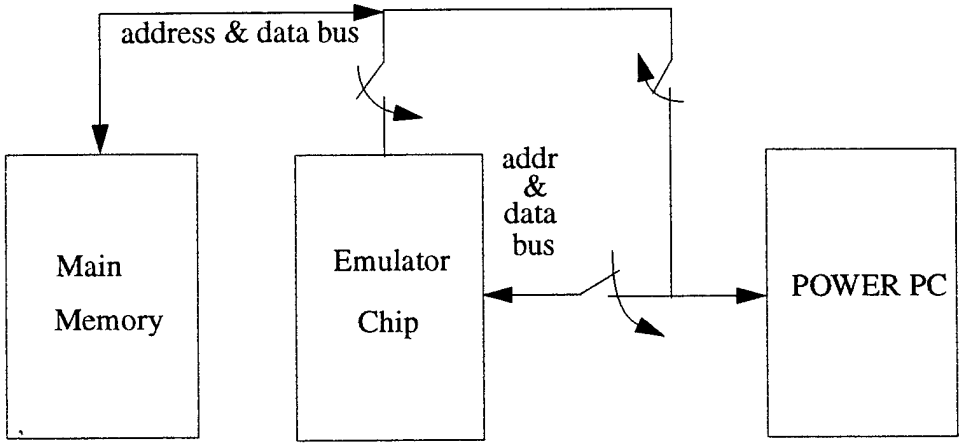

Figure 2 - The system overview of the pseudo emulator chip.

The two main functions of the emulator chip are to accept AN/AYK-14 instructions and data and then execute these instructions and data on a PowerPC microprocessor. The design of the emulator chip can be divided into two main sections according to their functions. The first section is the control section which functions similarly as a AN/AYK-14 processor. The second is the translation section which sends out a sequence of PowerPC instructions. The sequence of PowerPC instructions allows the PowerPC microprocessor to emulate the AN/AYK-14 correctly.

A simple block diagram of the emulator chip is shown in Figure 3. The control section includes the AN/AYK-14 instruction fetch and instruction decode units. The translation section includes the cycle counter, ROM containing the PowerPC instruction sequence, controller, 32 16-bit general purpose registers, logic unit, and PowerPC interface registers.

Because of the difference in the floating data format between the AN/AYK-14 and the PowerPC, the general registers which contain the AN/AYK-14 data format are required to stay in the emulator chip. The availability of the registers' content, and floating point data conversion circuit in the emulator chip reduces the floating point data conversion from 19 clock cycles to 1 clock cycle. The availability of the registers' content also speeds up the addressing mode decoding. The inclusion of the instruction decode units and general registers in the emulator chip reduces the cycle time requirement to decode the cascade addressing modes from 52 clock cycles to 5 clock cycles. The instruction decoder also speeds up the resetting of the carry and overflow bits according to AN/AYK-14 specification.

For every AN/AYK-14 executed instruction, the emulator chip automatically sends out the following PowerPC instructions:

6 LWA Ra,d(r0): load operand into register A
5 LHA Rm,d(r0): load operand into register M
4 ALU. Ra, Ra, Rm; execute e instruction
3 STW Ra, d(r0): store the data in register A to emulator chip
2 MFCD R31: move the conditional code to register 31
1 STW R31, d(r0) store the data in register 31 to emulator chip and update the conditional code

The emulator chip has a 5 cycle time overhead to execute each simple instruction such as AND, OR, XOR,..., These overhead instructions occupy a significant amount of the ROM's area, and complicate the translation control process. A logic unit which performs the AND, OR, XOR, can be easily designed and implemented in the emulator chip. The logic unit completes the instruction within a clock cycle, occupies less chip area and is a better design trade-off.

The operation of the emulator chip is summarized below: first, the instruction fetch requests the main memory to send an instruction. Upon receiving the instruction, the instruction decoder generates the operand address, and commands to direct the instruction execution. For a simple instruction, such as AND, the instructions are completed in the logic unit. For a complex instruction such as multiply or floating point arithmetic, the instruction decoder sends the command signal to the ROM containing the PowerPC instruction sequences, controller, and cycle counter. The ROM sends out the proper preprogrammed PowerPC instruction opcodes and the controller sends the correct data, addresses, register number, and register content to the PowerPC interface register. While ROM is sending data and opcodes to the PowerPC interface register, the cycle counter is starting to count down until it reaches a zero which indicates the end of the instruction sequence and signals the instruction fetch to request a new instruction.

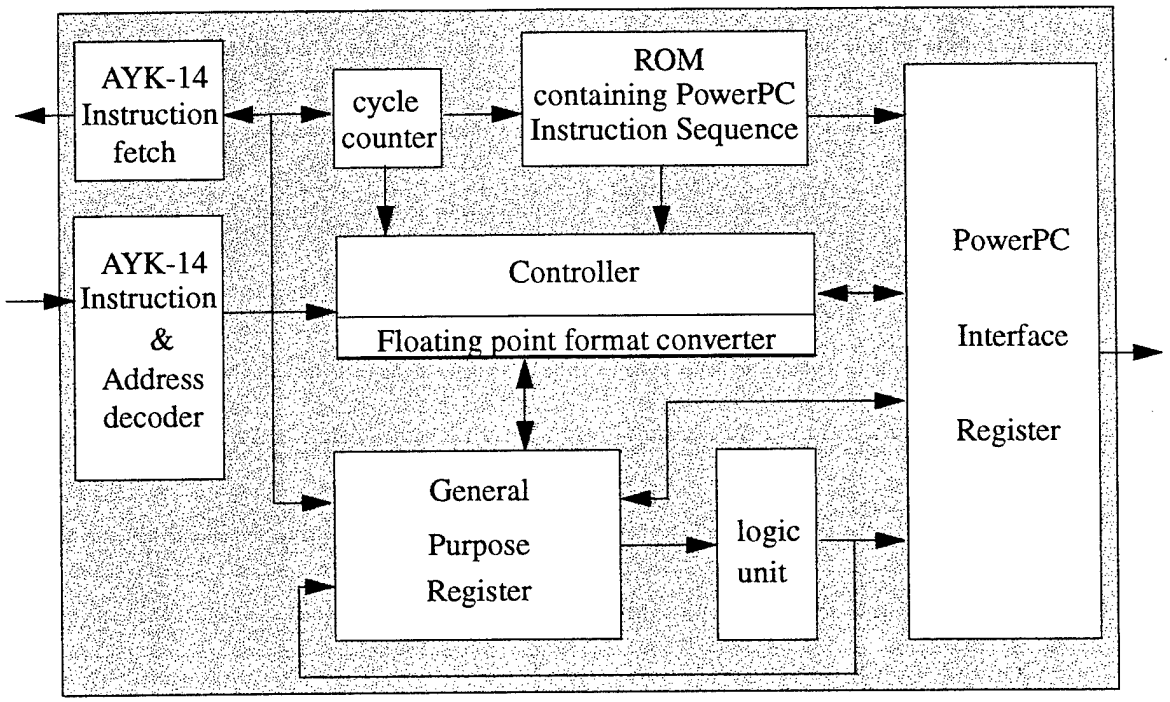

Figure 3 - The pseudo emulator chip block diagram.

1.3 Phase I accomplishments

1.3.1 Phase I Technical Objectives

SPEC proposed the following Phase I technical objectives.

- Develop the emulator chip logic which will allow for the AN/AYK-14 or MIL-STD-1750 Opcode to be executed on a commercially available 32-bit microprocessors such as Pentium, PowerPC, MIPS, or SPARC.
- Develop the circuit design for the emulator chip including ROM and random logic using HSPICE simulator.
- Develop a detailed layout for the ROM and random logic using SPEC's CMOS and GaAs CAD tools and cell libraries.
- Perform simulations of the emulator chip using Compass Design Automation's CAD tools and VHDL simulation.

SPEC has met all of the Phase I Program objectives. SPEC has analyzed and developed a complete design and SPEC has developed a VHDL based design. SPEC has improved the performance of the emulator chip by adding the address decoder, floating point conversion, and logic unit in the emulator chip.

SPEC has improved the chip density by laying out the folded ROM cell.

1.3.2 Phase I Task Results

Task 1: Develop Requirements and Specifications for the 32-Bit High Throughput Processor/Emulator Chip

SPEC will work jointly with the NAVY to review the range of commercially available 32-bit microprocessor architectures. SPEC and the NAVY will evaluate and select a candidate architecture for emulating the AN/AYK-14 or MIL-STD-1750 opcode. Each architecture will be evaluated for performance, chip size, power dissipation and the number of I/O pads.

Result: SPEC has developed a comprehensive design specification for the emulator chip which will direct the PowerPC to execute the AN/AYK-14 application software. The use of the PowerPC microprocessor to emulate the AN/AYK-14 is preferred by the NAVY. The emulator chip is designed based on AN/AYK-14(V) Programmer's Reference Manual (13211927D April 1995). This manual does not reveal the I/O connection. Therefore an estimate of the I/O requirements is used in the Phase I Program design of the emulator chip.

Task 2: Logic Design and Simulations

This task will include the design and simulation of the 32-bit high throughput emulator. The logic will be designed using SPEC's CMOS standard cell libraries for functional simulation, design verification and physical implementation.

Result: SPEC has developed a detailed VHDL ASIC design (see Appendix A). The VHDL ASIC design has been synthesized using a standard cell library to gate level. SPEC has also used a datapath compiler to design and implement the general register, logic unit, ROM, and PowerPC interface registers. SPEC has simulated the VHDL design to verify functional correctness.

Task 3: Circuit Design and Simulations

This task will include the design and simulation of the 32-bit high throughput processor/emulator using HSPICE. The circuits will include ROM and decoder. This task will also include optimization and trade-off studies for performance versus area, and performance versus power dissipation.

Result: SPEC has developed a detailed ROM design (see Section 2.4). The folded two transistor ROM memory cell which occupies minimum area is utilized in the design of the emulator chip. This ROM can be generated through the ROM compiler. The decoder can be generated through the logic synthesizer using the standard cell library. The ROM and standard cell library have been characterized by running HSPICE simulation.

Task 4: Physical Design and Verification

SPEC will layout the circuits that were designed in Task 3. SPEC will also place and route the logic designed in Task 2.

Result: SPEC has designed a custom layout of the memory cell to reduce the parasitic capacitance and size of the ROM. SPEC has manually placed the major components of the emulator chip to improve the connectivity between the emulator chip's components. The final routing is generated by running the Compass Design Automation routing tool.

Task 5: Phase II Development Plan and Final Report

SPEC will plan and direct tasks and schedules for program activities. The Principal Investigator will organize and conduct internal meetings, provide deliverables, and present briefings to management and Government Sponsors. A final comprehensive technical report will be prepared at the conclusion of the program. A detailed Program Plan, based upon the results of the Phase I research, will be developed for the Phase II Prototype Chip Program.

Result: The Phase II Program objective remains unchanged, and will include fabrication of a CMOS ASIC. SPEC has prepared a comprehensive final report describing the problems associated with the software or ROM based design, and the advantage and the design of the pseudo emulator chip.

2.0 ASIC Design

In order to ease the chip implementation, the emulator chip is partitioned into four physical design blocks: the random logic block, the 16-bit datapath block, the 32-bit datapath block, and the ROM memory block as described below.

2.1 The random logic block

The random logic block includes the instruction fetch, the instruction decoder, the address decoder, the floating point format converter, and the controller.

2.1.1 Instruction fetch

The instruction fetch, which requests the main memory to send out a new instruction, is composed of a program counter, an adder, and two 2-input muxes. The logic diagram of the program counter is shown in Figure 4. The content of the program counter depends on the instruction executed. When the emulator executes a single instruction, the program counter equals the program counter plus one. When the emulator executes a double instruction, the program counter equals the program counter plus two. When the emulator executes a jump or a branch, the program counter equals the new program counter.

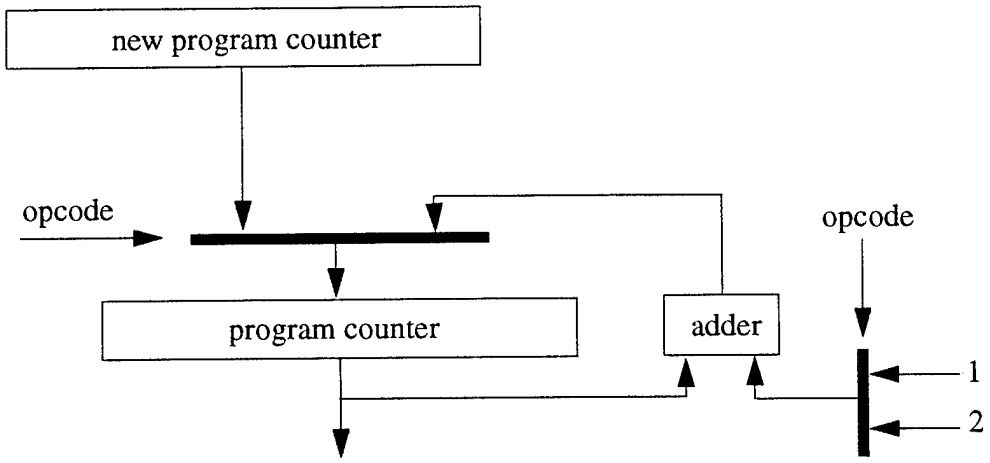

Figure 4 - The logic diagram of the program counter.

2.1.2 Instruction decoder.

The instruction decoder accepts N-bits of the instruction opcode and generate 2^N control responses. The implementation of this decoder is straight forward. The instruction decoder also controls the state machine of the emulator chip. The instruction decoder sends out the number of clock cycles required to complete the emulation.

2.1.3 Instruction address decoder.

Because of the uniqueness of the AN/AYK-14 addressing mode, an address decoder is required to speed up the emulator process. The logic diagram of the address decoder is shown in Figure 5

The longest addressing mode requires 5 clock cycles. The first cycle is used to decode the instructions and register numbers. The second cycle is used to generate the address for IW1. The third and fourth cycles are used to receive the IW1 and IW2 operands and generate the indirect address IW2+d. The final address Y which equals the content of the address IW2+d is ready at the end of the fifth cycle.

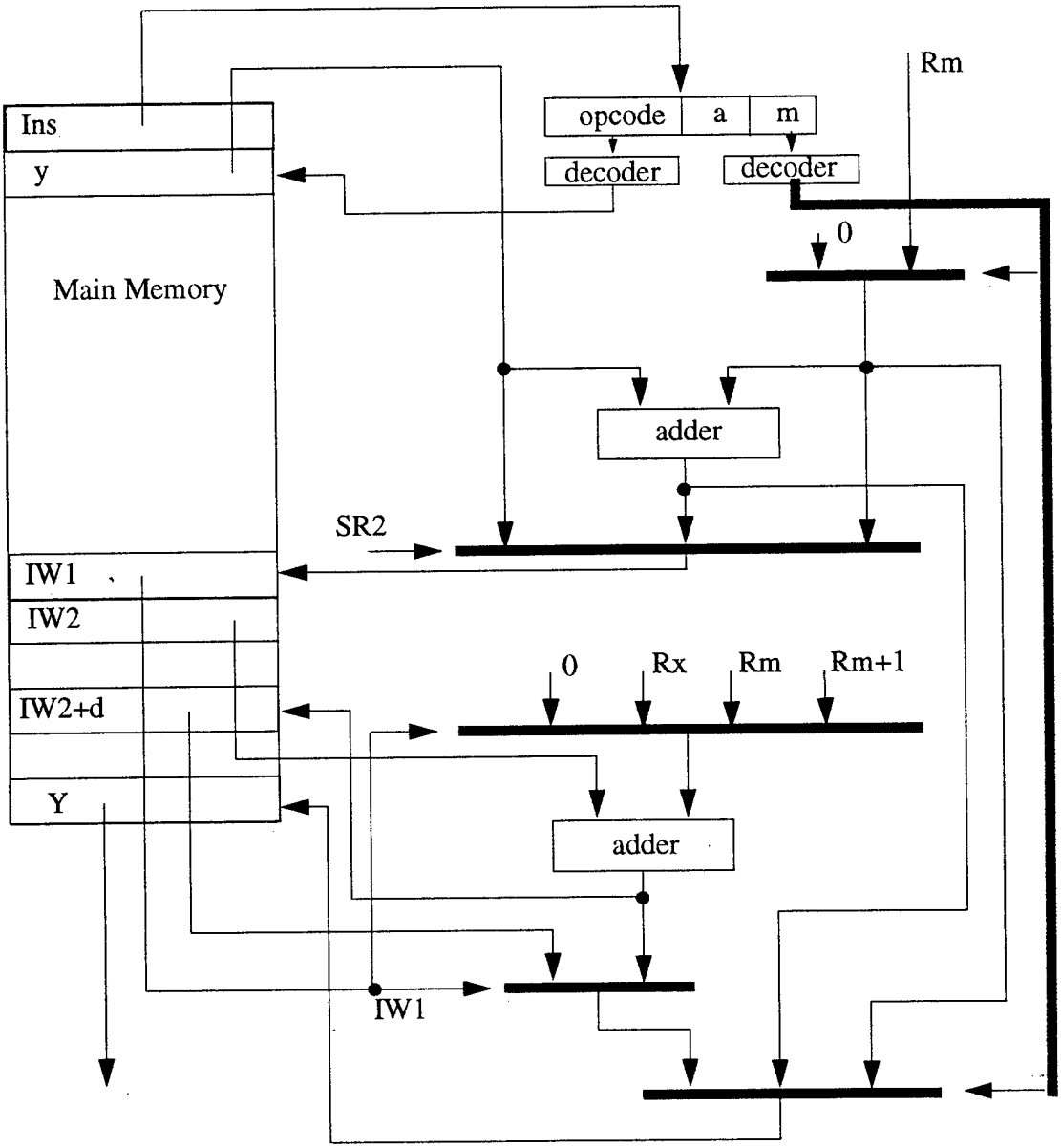

Figure 5 - The logic diagram of the address decoder

2.1.4 Floating Point Format conversion.

A conversion circuit is required to convert the AN/AYK-14 floating point format to PowerPC double precision format and vice versa. The block diagram of the logic implementation is shown in Figure 6.

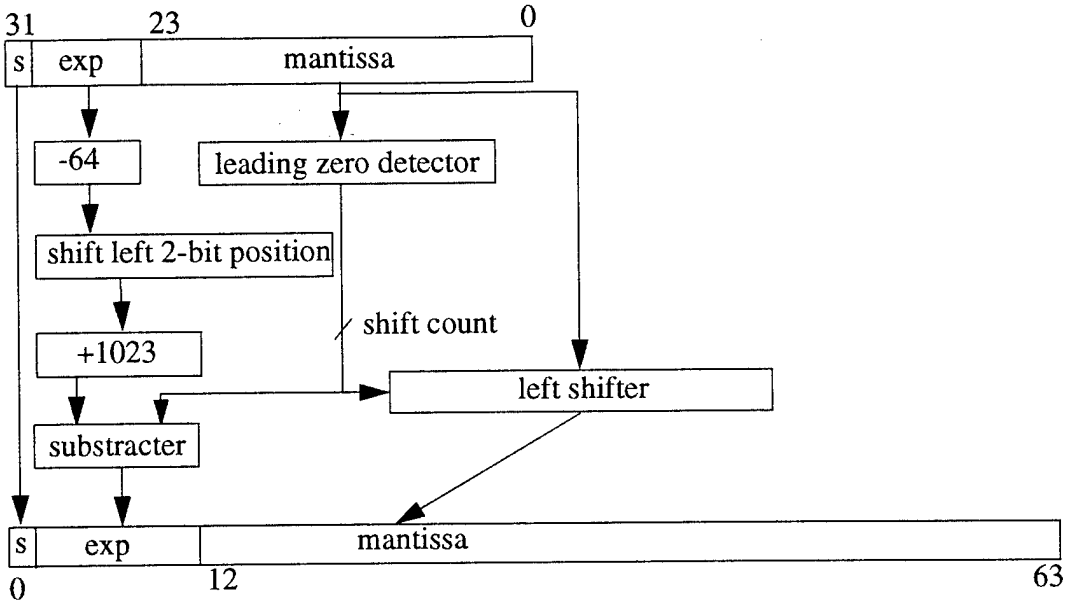

Figure 6 - The logic diagram of the floating point format conversion circuit.

2.1.5 Controller.

The function of the controller is to direct the ROM to send out the instructions while the general register sends out data to the PowerPC interface register. Upon receiving an instruction execute, such as multiply, the instruction address generator sends out the corresponding instruction address for the ROM, and the number of cycles required to complete the emulation. According to the instruction address, the ROM sends out the instruction to the PowerPC interface registers. The instruction address is increased by one, and the number of the cycles required to complete the emulation reduces by one. This process continues until the required cycles equals zero. When the required cycles equals zero, the instruction address receives a new address from the instruction address and the same process starts over again. The block diagram of the controller is shown in Figure 7.

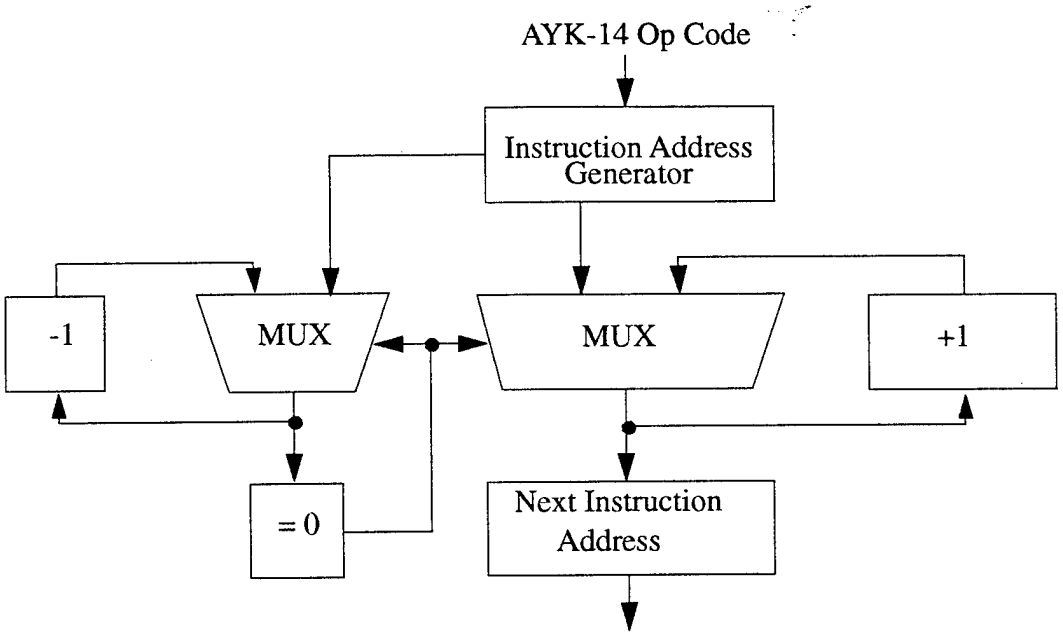

Figure 7 - The logic diagram for the controller

This logic block is designed using VHDL to describe the logic function. A logic synthesizer is required to convert the VHDL to a standard cell level netlist. This netlist is placed and routed to form the physical layout.

2.2 The 16-bit datapath block

The 16-bit datapath block includes 32 16-bit general registers, a logic unit, and a zero detector. The general registers are composed of two read, and one write memory cells. The implementation of the logic block which executes two input AND, OR and EXCUSIVE OR logic functions, and the zero detector occupy a minimum amount of the silicon area. However, it requires only one clock cycle to complete the execution. Whereas, the emulation of this simple function requires 5 clock cycles. The implementation of the logic unit in the emulator chip is a good trade-off between area and performance. The zero detector is required to update the conditional code.

The general registers and the logic unit can be implemented through the datapath compiler. The logic schematic of the 16-bit datapath block is shown in Figure 8.

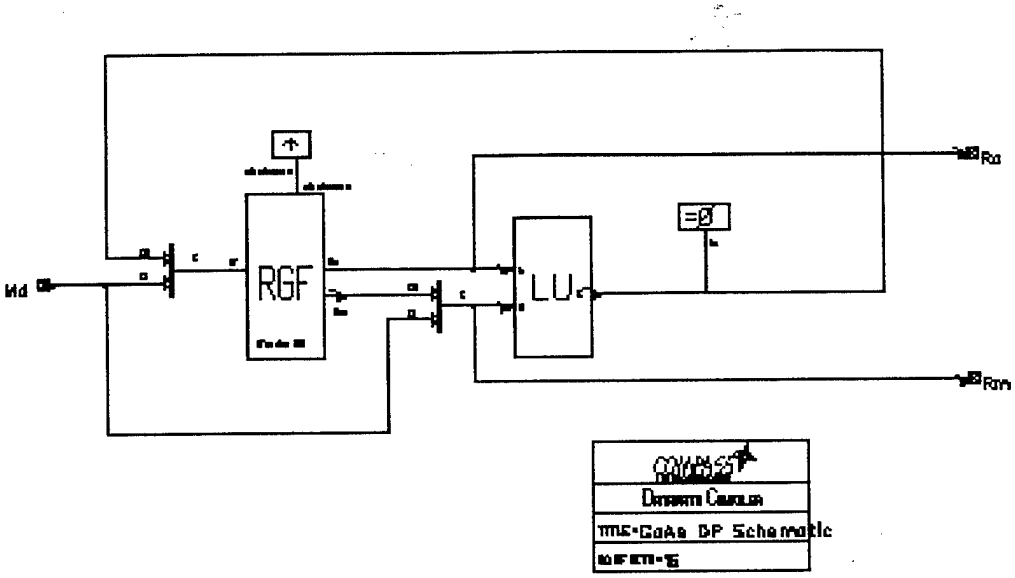

Figure 8 - The logic schematic of the 16-bit datapath block

2.3 The 32-bit datapath block

The PowerPC interface register is an array of 32 32-bit registers. The PowerPC interface registers have two read ports and two write ports, which allow a simultaneously read from 2 registers or write into 2 registers. Depending on the select signals, the registers store the data from the general register, the main memory, the floating point conversion data, or the PowerPC. The interface register can either store the 16-bit data from main memory or register file at the high 16-bit positions or at the low 16-bit position.The two read ports are connected to the output drivers.

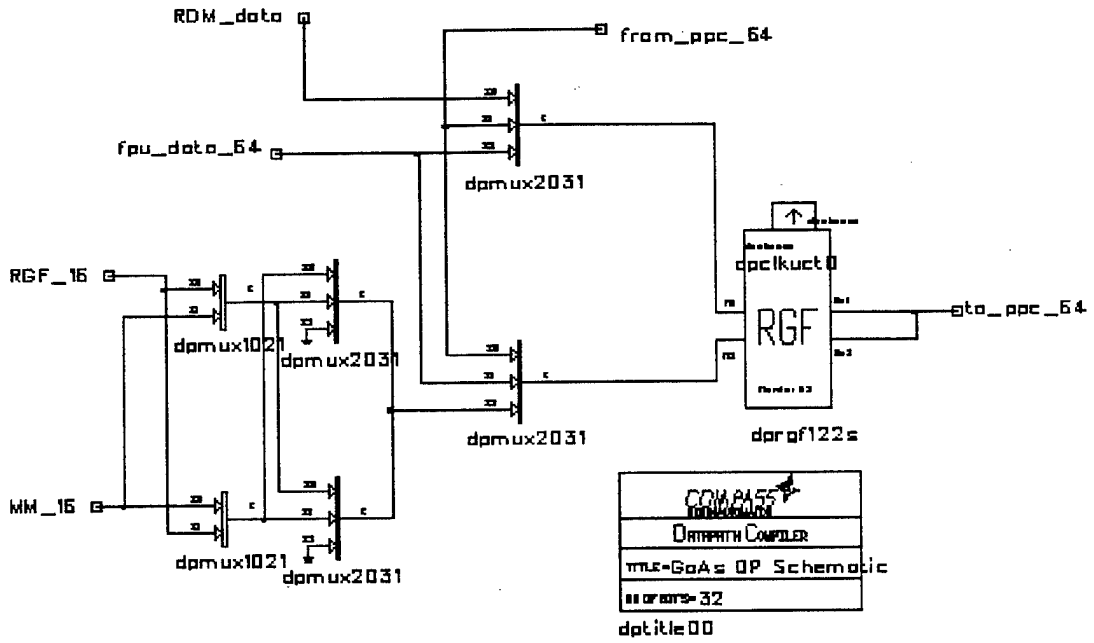

Figure 9 - The logic schematic of the 32-bit datapath block.

2.4 The ROM memory block

The emulator chip requires a ROM to store the preprogrammed PowerPC instructions. The transistor schematic and the layout of the ROM cell is shown in Figures 10 and 11. To store a one, a via is dropped between the VDD bus and diffusion. Similarly to store a zero, a via is dropped between VSS bus and diffusion. The process of storing ROM data can be done automatically by the ROM complier program.

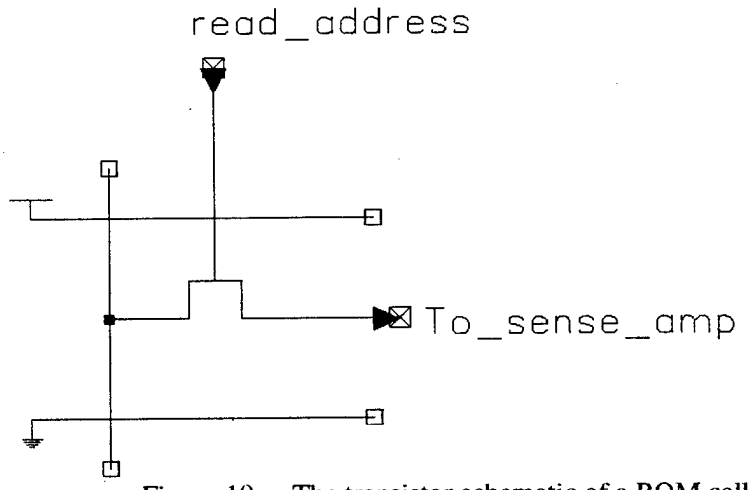

Figure 10 - The transistor schematic of a ROM cell.

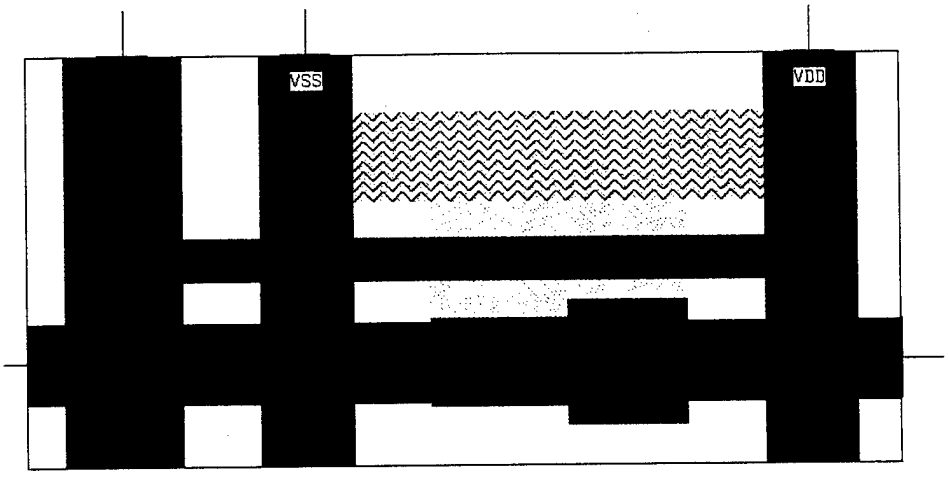

Figure 11 - The layout of a ROM cell

SPEC has laid out a new ROM cell by folding two single transistor ROM cells into a double transistor cell. The new ROM cell has better performance and higher density because it occupies less area and has less parasitic capacitance. The transistor schematic and layout of the folded ROM cell is shown in Figures 12 and 13.

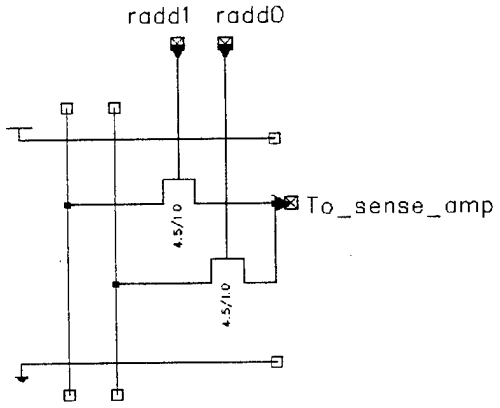

Figure 12 - The schematic of a folded ROM cell.

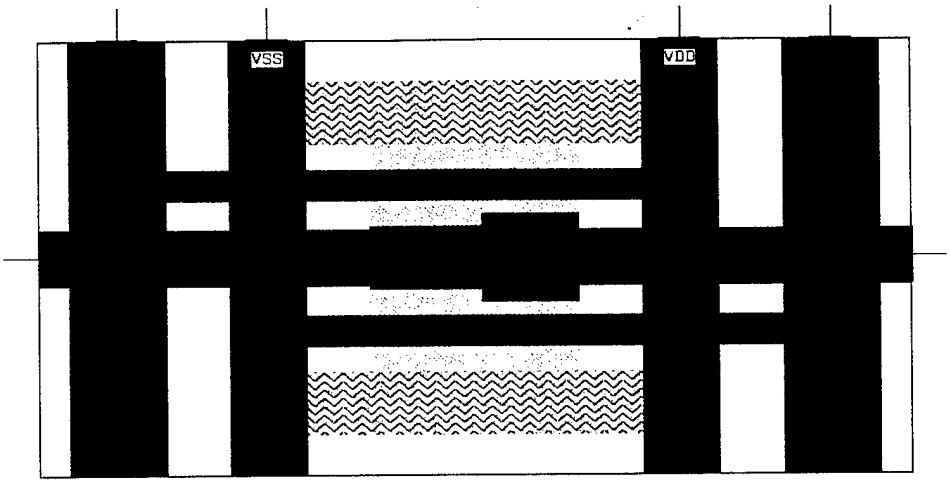

Figure 13 - The layout of a folded ROM cell

An array of 18X17 ROM cells is complied and demonstrated in Figure 14. A larger ROM is used in the emulator chip to store the preprogrammed PowerPC instruction sequences.

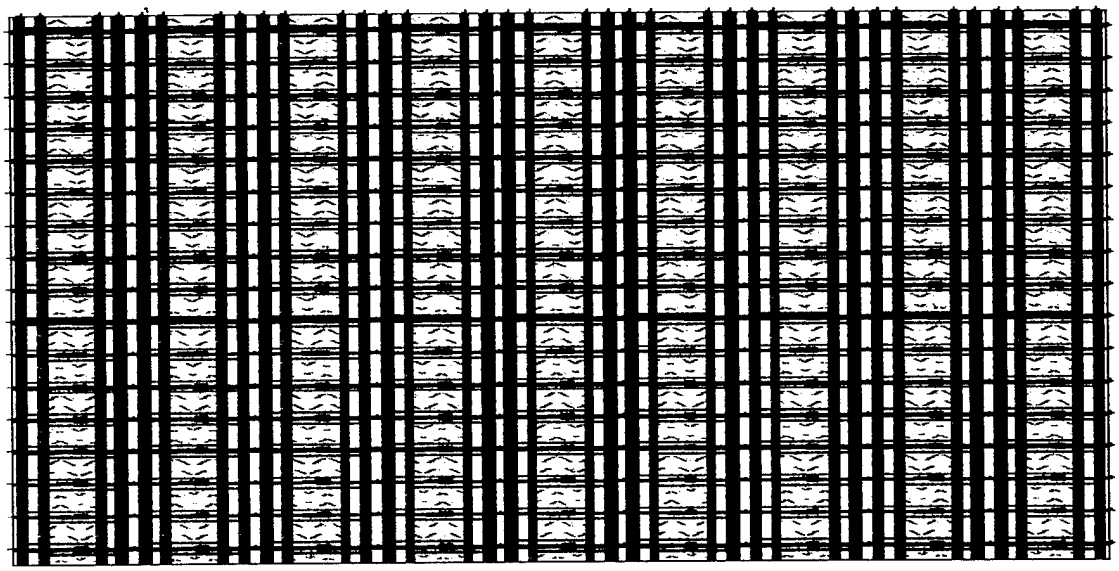

Figure 14 - An array of 18X17 ROM

3.0 Pseudo Emulator Chip.

The top level (or chip level) schematic of the pseudo emulator chip is required for the chip compiler to place and route the random logic block, the ROM, the 16-bit datapath block, and the 32-bit datapath block. The external input and output signals of these blocks are connected with the receiver and driver pads of the chip. The power supply pads are also provided to support the power and ground voltages. The external signals are derived from the NTDS channel interface configuration.

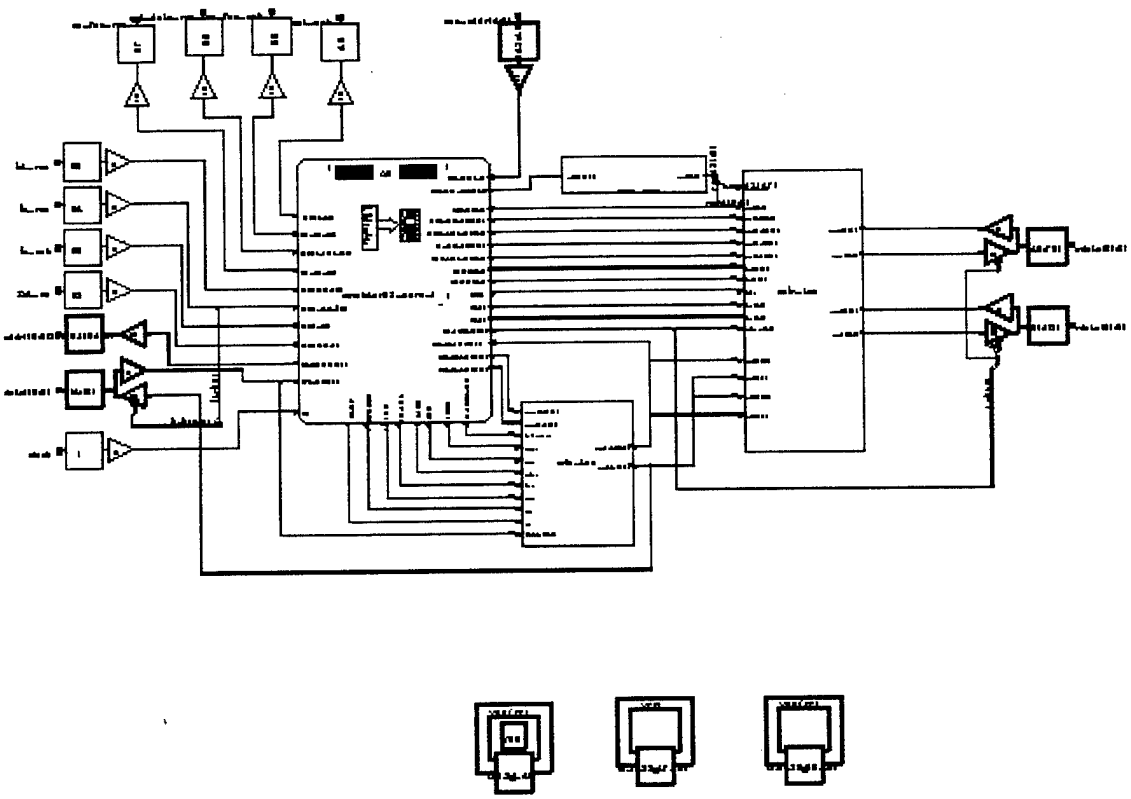

Figure 15 - The top level logic schematic of the pseudo emulator chip.

The physical layout of the pseudo emulator chip is completed and is shown in Figure 16. The random logic block, the ROM, the 16-bit datapath block, and 32-bit datapath are easily identified. The chip has some vacant space for additional I/O and power supply pads.

SPEC has demonstrated an unique design approach which provides direct execution of existing AN/ AYK-14 instructions on a PowerPC.

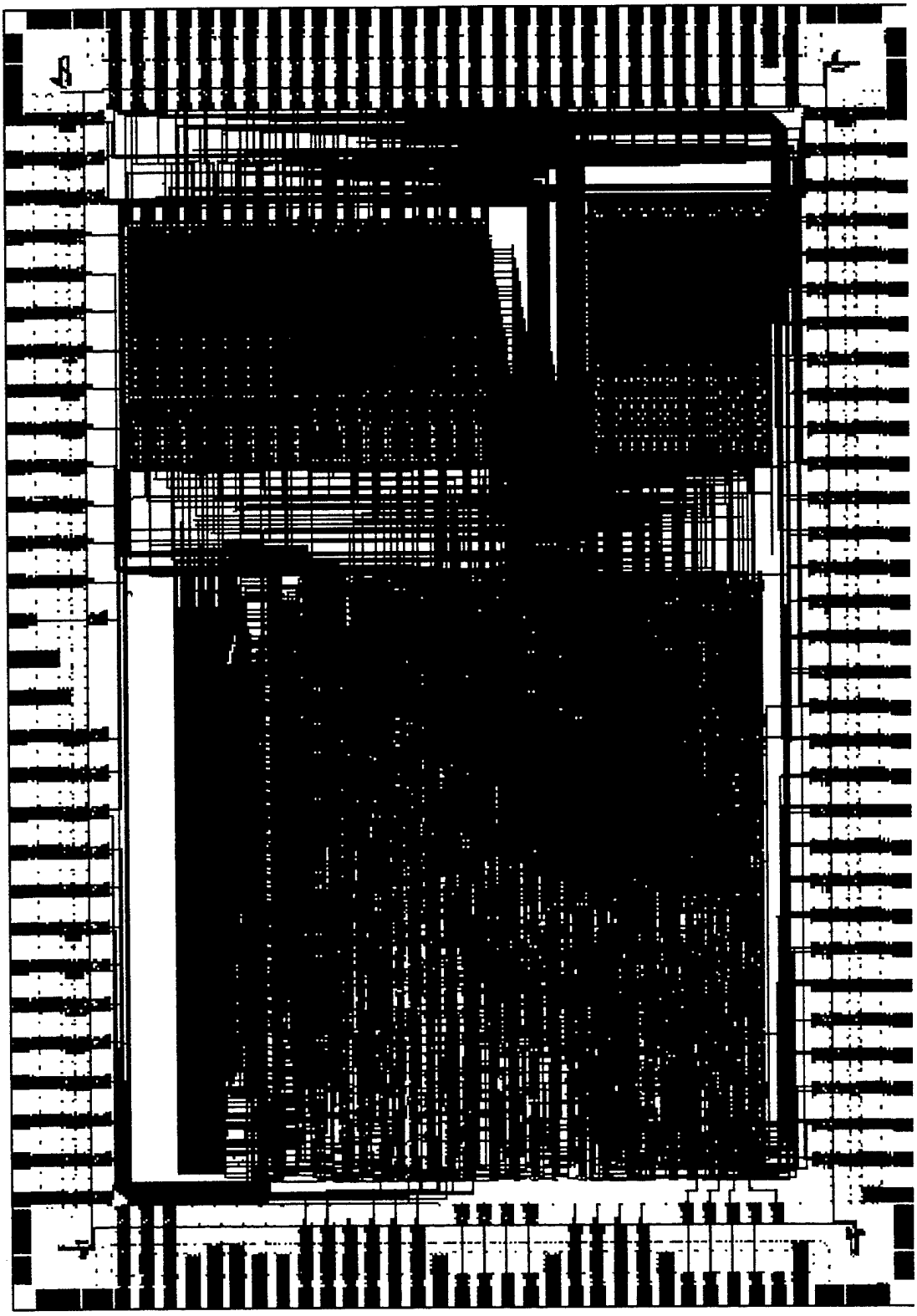

Figure 16 - Chip Design

4.0 VHDL Simulation.

Because of the unavailability of the PowerPC VHDL model in the Phase I Program, the emulation system can not be simulated at the system level. Therefore, the VHDL simulation of the pseudo emulator is carried out by itself.

The following is a description of how the 16 bit AN/AYK-14 instruction set is decoded into the 32 bit PowerPC instruction format. To emulate an AYK "Add" instruction, the pseudo emulator chip sends out 7 PowerPC instructions in a sequence to the PowerPC processor.

AYK	PowerPC instruction (PPC)	32 bit register's content
4b a m	LWZ Ra, d(r0)	32 A 0 d
	LWZ Rm, d(r0)	32 M 0 d+1
	ADDCO Ra, Ra, Rm	31 A A M 1 10 1
	STW Ra, d(r0)	36 A 0 d
	MCRXR CRFD	31 1 0 0 512 0
	MFCD R31	31 31 0 0 19 0
	STW Rs, d(r0)	31 31 0 d+1

where 4b = AN/AYK-14 op code

32,31,36 = PPC op code

512, 19 = PPC extend op code

A = a

M = m

d = data address

AYK 16 bit instruction format : bit 15 downto 8 is opcode

bit 7 downto 4 is a operand

bit 3 downto 0 is m operand

PPC 32 bit instruction format : bit 31 downto 21 is data addr

bit 20 downto 16 is B operand

bit 15 downto 11 is A operand

bit 10 downto 6 is D operand

bit 5 downto 0 is op code

The VHDL simulation results are shown in the waveform diagram (Figure 17). Two AYK instructions, "ADD" and "MULTIPLY", are used to demonstrate the simulation results.

ADD instruction input = "0100101101001000"

Multiply instruction input = "0101100101001100"

The simulation starts out with reading the input instruction and assigns the initial value to the gv_cycle_counter, the gv_reg32_addr, and the gv_data_addr. The gv_cycle_counter is a decrementer. It starts with the total number of instructions required to emulate and counts down to zero. The gv_reg32_addr and gv_data_addr contain the PowerPC interface register addresses. The gv_reg32_addr is initially set at zero and gv_data_addr starts out at ten. The gv_reg32_addr will increment by one every time it writes the PowerPC instruction into the interface register. Similarly, the gv_data_addr will increment by one every time it writes the data into the interface register. The pc_int_reg_content stores the actual PowerPC instruction sequences.

So for the ADD instruction, the gv_cycle_counter starts out with 7. The first 32 bit PowerPC instruction sequence to be written is

bit 0 to 5 is op code = 32 (decimal) = 100000

bit 6 to 10 is D operand = A (hex) = 00100

bit 11 to 15 is A operand = 0 = 00000

bit 16 to 20 is B operand = 0 = 00000

bit 21 to 31 is data address = 10 (decimal) = 00000001010

pc_int_reg_content(0) = "10000000010000000000000001010"

The next PowerPC instruction sequence starts with gv_cycle_counter = 6, gv_reg32_addr = 1 and gv_data_addr = 11. The PowerPC interface register content is

bit 0 to 5 is op code = 32 (decimal) = 100000

bit 6 to 10 is D operand = M (hex) = 01000

bit 11 to 15 is A operand = 0 = 00000

bit 16 to 20 is B operand = 0 = 00000

bit 21 to 31 is data address = 11 (decimal) = 00000001011

pc_int_reg_content(1) = "10000001000000000000000001011"

The third PPC instruction sequence starts with gv_cycle_counter = 5 and gv_reg32_addr = 2. The PPC interface register content is

bit 0 to 5 is op code = 31 (decimal) = 011111

bit 6 to 10 is D operand = A (hex) = 00100

bit 11 to 15 is A operand	= A	= 00100
bit 16 to 20 is B operand	= M	= 01000
bit 21		= 1
bit 22 to 30 is data address	= 10	= 00000101
bit 31		= 1

pc_int_reg_content(2) = "01111100100001000100010000001011"

The rest of the instruction sequences are generated the same way as the three instructions above. The emulating process will stop until gv_cycle_counter signal reaches the last instruction. The gv_cycle_counter, gv_reg32_addr, and gv_data_addr will reset themselves to a new values when reading the next instruction inputs.

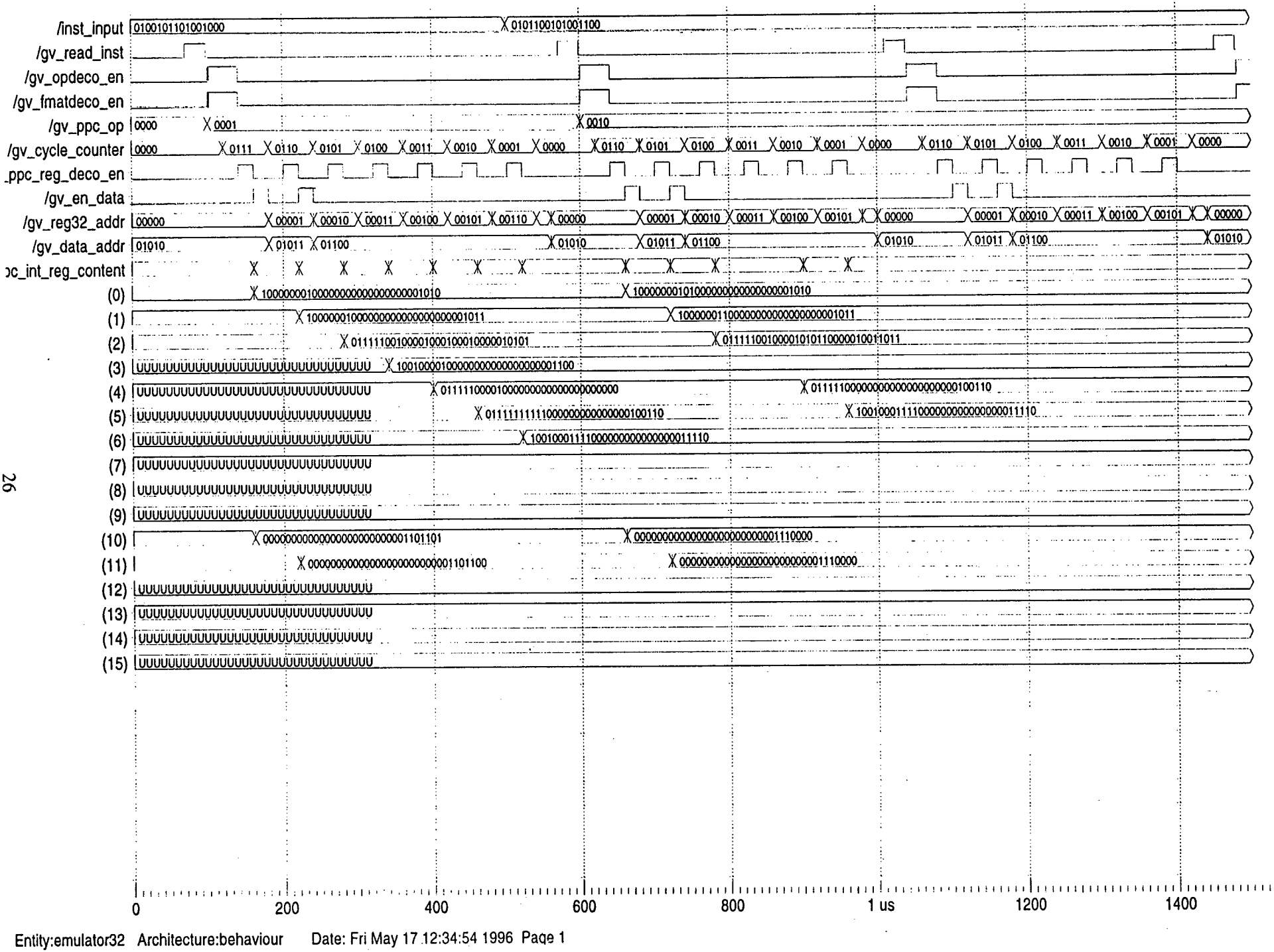

Figure 17 The wave form of the VHDL simulation.

5.0 Phase II Program Plan

SPEC proposes to perform the tasks described below as part of the 32-Bit Emulator Phase II Program.

Task 1 - Finalize Emulator and PowerPC Design Requirements

SPEC will review the existing emulator design and current AN/AYK-14 implementations with the Government. This will result in determination of required emulator chip operating speeds and interfaces. SPEC will assess future processing needs to determine what PowerPC functionality will be needed as the designs migrate from legacy AN/AYK-14 code to PowerPC code. SPEC will also assess the interfaces required on the replacement 32-Bit Emulator processor card. SPEC will develop a 32-Bit Emulator circuit card assembly specification and a 32-Bit Emulator Chip specification. This task will also determine in which technology the 32-Bit Emulator Chip will be implemented (e.g. GaAs or CMOS).

Task 2 - Emulator Chip Design Optimization

SPEC will perform revisions to the Phase I chip design to accommodate changes and/or improvements required to meet the Task 1 specification. SPEC will perform a layout of the chip design for fabrication.

Task 3 - Fabricate Emulator Chip

SPEC will subcontract for fabrication of the 32-Bit Emulator Chip. The chip will be fabricated either in GaAs or CMOS, as determined in Task 1. SPEC will assess the anticipated yield, and fabricate a sufficient number of chips to yield a minimum of two units.

Task 4 - AN/AYK-14 Circuit Card Assembly Design

SPEC will design an AN/AYK-14 Circuit Card Assembly (CCA). The CCA will contain the 32-Bit Emulator Chip, a PowerPC processor, and the standard AN/AYK-14 processor card interfaces. The CCA will be designed to directly replace the current AN/AYK-14 processor card.

Task 5 - AN/AYK-14 CCA Fabrication and Test

SPEC will fabricate an AN/AYK-14 CCA. The CCA will be tested in an AN/AYK-14 to verify hardware functionality and basic software emulation.

Task 6 - Performance Validation Testing

SPEC will develop a test plan to verify the performance of the 32-Bit Emulator Chip. Testing will verify proper operation of all AN/AYK-14 instructions. In addition, testing will verify the bypass mode operation and interface control.

Task 7 - Prepare Quarterly and Final Reports

SPEC will prepare quarterly reports detailing the results and accomplishments against each task during the reporting period, the work planned for the coming period, and any problems encountered during the course of the program. SPEC will prepare a comprehensive final report detailing the results of the program, including a detailed description of the emulator chip, circuit card assembly, and test results.

The proposed program schedule is shown in Figure 18.

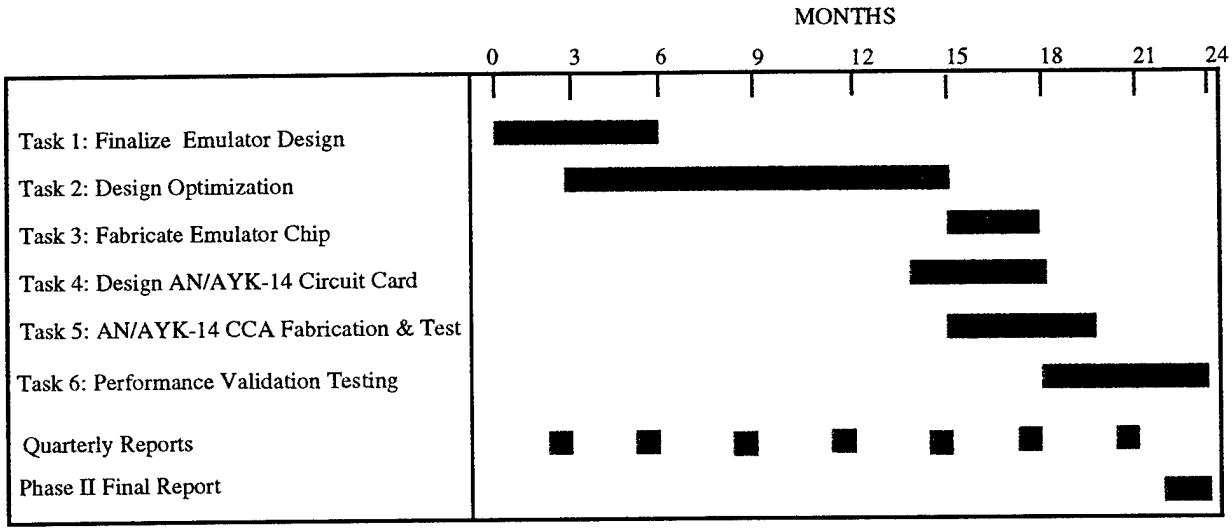

Figure 18 - Program Schedule

Appendix A - VHDL Code

```
-- ***************************************************
-- This vhdl code is the Main module.
-- It will read the opcode and output the result
--
-- ***************************************************

--USE std.std_logic.ALL;
--USE std.std.ttl.ALL;
library IEEE;
use IEEE.std_logic_1164.all;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.all;
--compass compile_off
use COMPASS_LIB.COMPASS_ETC.ALL;
--compass compile_on
use Work.Emulator_datatype.all ;


ENTITY emulator32 IS
 --  generic (Tpd : Time := unit_delay);

   PORT (
          ayk_ins          : IN std_logic_vector(15 downto 0);
          input_data_req   : IN std_logic ; -- ready
          interrupt_req    : IN std_logic ;
          output_data_req  : IN std_logic ;
          ext_fun_req      : IN std_logic ;
          clk              : IN std_logic ;
          reg_rm_data      : IN std_logic_vector(15 downto 0);
          rom_op           : IN std_logic_vector(16 downto 0);
          ppc_addr         : IN  std_logic_vector(4 downto 0);
          carry            : IN  std_logic;
          overflow         : IN  std_logic;
          zero             : IN  std_logic;

          pc_counter       : OUT std_logic_vector(15 downto 0) ;
          ext_fun_ack      : OUT std_logic ;
          output_ack       : OUT std_logic ; -- write
          interrupt_en     : OUT std_logic;
          input_ack        : OUT std_logic ; -- read
          reg_rm_addr      : OUT std_logic_vector(31 downto 0);
          reg_ra_addr      : OUT std_logic_vector(31 downto 0);
          sel_data         : OUT std_logic ;
          sel_logic_mem    : OUT std_logic ;
          fpins            : OUT std_logic;
          orop             : OUT std_logic;
          andop            : OUT std_logic;
          xorop            : OUT std_logic;
          sel_signalB      : OUT std_logic_vector(1 downto 0);
          A                : OUT std_logic_vector(4 downto 0);
          M                : OUT std_logic_vector(4 downto 0);
          rom_addr_dec     : OUT std_logic_vector(299 downto 0);
          fpdout0          : OUT  std_logic_vector(31 downto 0);
          fpdout1          : OUT  std_logic_vector(31 downto 0);
          wppc_addr_dec    : OUT  std_logic_vector(31 downto 0);
          rppc_addr_dec    : OUT  std_logic_vector(31 downto 0);
          wd_addr_dec      : OUT  std_logic_vector(31 downto 0);
          wpcir_addr_dec   : OUT  std_logic_vector(31 downto 0));

END emulator32 ;

 --          result          : OUT std_logic_vector(15 downto 0)) ;
 --         SR1data     : INOUT bit_vector(15 downto 0);
```



```
 --         SR2data     : INOUT std_logic_vector(15 downto 0);
 --         ydata       : IN std_logic_vector(15 downto 0);
 --        Pdata       : IN std_logic_vector(15 downto 0);
 --          IW1data     : IN std_logic_vector(15 downto 0);
 --          IW2data     : IN std_logic_vector(15 downto 0);

ARCHITECTURE behaviour OF emulator32 IS
--compass dp_gates;

-- >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
-- >>>>>>>>>>>>>  Define Component <<<<<<<<<<<<<<<<<<<<<<<<
-- >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

COMPONENT pccon
   PORT (Inputdata : IN std_logic_vector(15 downto 10);
         m         : IN std_logic_vector(3 downto 0);
         opdeco_en : IN std_logic ;
         sins      : OUT std_logic;
         dins      : OUT std_logic;
         bins      : OUT std_logic;
         fpins      : OUT std_logic ;
         orop       : OUT std_logic ;
         andop      : OUT std_logic ;
         xorop      : OUT std_logic ;
         sr1op     : OUT std_logic;
         sr2op     : OUT std_logic;
         selmr     : OUT std_logic);
  end component ;


 component em_ctlf
    port ( ayk_ins : in Std_Logic_Vector(15 downto 0);
           rom_addr : out Std_Logic_Vector(8 downto 0);
           CLK : in Std_Logic ;
           new_ppc_ins : out Std_Logic );
end component ;

component fmatdeco
   PORT (inputdata   : IN std_logic_vector(15 downto 0);
         SR114     : IN std_logic                      ;
         SR2hb     : IN std_logic_vector(7  downto 0);
         ydata     : IN std_logic_vector(15 downto 0);
         npcount   : IN std_logic_vector(15 downto 0);
         IW1data   : IN std_logic_vector(15 downto 0);
         IW2data   : IN std_logic_vector(15 downto 0);
         IW2ddata   : IN std_logic_vector(15 downto 0);
         ayk_dcyl  : IN std_logic ;
         ayk_ycyl  : IN std_logic ;
         ayk_w1cyl : IN std_logic ;
         ayk_w2cyl : IN std_logic ;
         ayk_w2dcyl : IN std_logic ;
         ayk_ecyl : IN std_logic ;
         clk        : IN std_logic ;
         sins       : IN std_logic ;
         dins       : IN std_logic ;
         bins       : IN std_logic ;
         rm_data   : IN   std_logic_vector(15 downto 0);
         odd       : OUT std_logic;
         ra_addr   : OUT std_logic_vector(31 downto 0);
         rm_addr   : OUT std_logic_vector(31 downto 0);
         a_addr    : OUT std_logic_vector(4  downto 0);
         m_addr    : OUT std_logic_vector(4  downto 0);
         mem_addr   : OUT std_logic_vector(15 downto 0));
END component  ;
```

```
COMPONENT em_sm
    port ( ayk_ins : in Std_Logic_Vector(15 downto 0) ;
           clk      : in  std_logic;
           sr1op    : in  std_logic;
           sr2op    : in  std_logic;
           inputdata: out std_logic_vector(15 downto 0);
           ydata: out std_logic_vector(15 downto 0);
           iw1data: out std_logic_vector(15 downto 0);
           iw2data: out std_logic_vector(15 downto 0);
           sr114  : out std_logic                    ;
           sr2hb  : out std_logic_vector(7  downto 0);
           edata  : out std_logic_vector(15 downto 0);
           rr     : out std_logic ;
           ayk_dcyl : out std_logic;
           ayk_ecyl : out std_logic;
           ayk_w1cyl : out std_logic;
           ayk_w2cyl : out std_logic;
           ayk_w2dcyl : out std_logic;
           ayk_ycyl : out std_logic);
END COMPONENT ;

component romdeco
   PORT (rominput  : IN std_logic_vector(8 downto 0);
         rom_addr  : OUT std_logic_vector(299 downto 0)) ;
END component ;

component fpoint
PORT (edata: IN  std_logic_vector(15 downto 0);
      Rgfdata: IN  std_logic_vector(15 downto 0);
      RR: IN  std_logic;
      odd: IN  std_logic;
      fpdout0: OUT  std_logic_vector(31 downto 0);
      fpdout1: OUT  std_logic_vector(31 downto 0));
END component ;

component  em_intf
    port ( rom_op : in Std_Logic_Vector(16 downto 0);
           ppc_addr  : in  Std_Logic_Vector(4 downto 0);
           new_ppc_ins : in  Std_Logic;
           wppc_addr_dec : out Std_Logic_Vector(31 downto 0);
           rppc_addr_dec : out Std_Logic_Vector(31 downto 0);
           wd_addr_dec : out Std_Logic_Vector(31 downto 0);
           wpcir_addr_dec : out Std_Logic_Vector(31 downto 0);
           CLK : in Std_Logic );
end component;

    signal   inputdata : std_logic_vector(15 downto 0);
    signal   ydata : std_logic_vector(15 downto 0);
    signal   iw1data : std_logic_vector(15 downto 0);
    signal   iw2data : std_logic_vector(15 downto 0);
    signal   sr114: std_logic ;
    signal   sr2hb   : std_logic_vector(7  downto 0);
    signal   edata : std_logic_vector(15 downto 0);
    signal   rr    : std_logic ;
    signal   odd   : std_logic ;
    signal   sr1op : std_logic ;
    signal   sr2op : std_logic;
    signal   ayk_dcyl:std_logic ;
    signal   ayk_ecyl:std_logic ;
    signal   ayk_w1cyl:std_logic;
    signal   ayk_w2cyl:std_logic;
    signal   ayk_w2dcyl:std_logic;
    signal   ayk_ycyl:std_logic;
```



```
signal   npcount  : std_logic_vector(15 downto 0) ;
signal        IW2ddata : std_logic_vector(15 downto 0) ;
signal        sins     :std_logic ;
signal        dins     :std_logic ;
signal        bins     :std_logic ;
signal     new_ppc_ins  :std_logic ;
signal        rm_data   : std_logic_vector(15 downto 0) ;
signal        rom_addr   : std_logic_vector(8 downto 0) ;

-- $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
BEGIN -- behaviour begin
-- $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


state_machine : em_sm
   port map( ayk_ins,
             clk ,
             sr1op,
             sr2op ,
             inputdata,
             ydata,
             iw1data,
             iw2data,
             sr114  ,
             sr2hb  ,
             edata  ,
             RR     ,
             ayk_dcyl ,
             ayk_ecyl ,
             ayk_w1cyl,
             ayk_w2cyl,
             ayk_w2dcyl,
             ayk_ycyl);

address_decoder : fmatdeco
   PORT map(inputdata,
         SR114    ,
         SR2hb    ,
         ydata    ,
         npcount  ,
         IW1data  ,
         IW2data  ,
         IW2ddata ,
         ayk_dcyl ,
         ayk_ycyl ,
         ayk_w1cyl ,
         ayk_w2cyl ,
         ayk_w2dcyl,
         ayk_ecyl  ,
         clk       ,
         sins      ,
         dins      ,
         bins      ,
         rm_data   ,
         odd       ,
         reg_ra_addr   ,
         reg_rm_addr   ,
         A         ,
         M         ,
         pc_counter );

 controller :  em_ctlf
    port map( inputdata,
          rom_addr ,
```

```
          CLK  ,
          new_ppc_ins  );

pc_control : pccon
   PORT map(Inputdata(15 downto 10) ,
         inputdata(3 downto 0) ,
         ayk_dcyl  ,
         sins      ,
         dins      ,
         bins      ,
         fpins     ,
         orop      ,
         andop     ,
         xorop     ,
         sr1op     ,
         sr2op     ,
         sel_data     );

romdecoder : romdeco
   PORT map(rom_addr,
         rom_addr_dec) ;

flpconver : fpoint
PORT map(edata,
      reg_Rm_data,
      RR,
      odd,
      fpdout0,
      fpdout1 );

interface :  em_intf
    port map( rom_op,
          ppc_addr,
          new_ppc_ins,
          wppc_addr_dec,
          rppc_addr_dec,
          wd_addr_dec,
          wpcir_addr_dec,
          CLK );

END behaviour;
```

```
--USE std.std_logic.ALL;
--USE std.std.ttl.ALL;
library IEEE;
use IEEE.std_logic_1164.all;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.all;
-- use COMPASS_LIB.COMPASS_ETC.all;

PACKAGE Emulator_datatype IS

  constant unit_delay : Time := 1 ns;

  subtype bit_32 is bit_vector(31 downto 0);
  type bit_32_array is array(integer range <>) of bit_32;

 -- function resolve_bit_32(driver : in bit_32_array)return bit_32;
 -- subtype bus_bit_32 is resolve_bit_32 bit_32 ;

  subtype reg_addr is natural range 0 to 32;
  type register32_array is array(reg_addr) of std_logic_vector(0 to 31);

  subtype bits_16 is bit_vector(15 downto 0);
  type bits_16_array is array(integer range <>) of bits_16 ;

-- function resolve_bits_16(driver : in bits_16_array) return bits_16 ;
--  subtype bus_bits_16 is resolve_bits_16 bits_16 ;

  subtype bit_16 is std_logic_vector(15 downto 0);
  subtype bit_8 is bit_vector(7 downto 0);
  subtype bit_5 is std_logic_vector(0 to 4);
  subtype bit_6 is std_logic_vector(0 to 5);


  function bits_to_int(bits: in bit_vector) return integer;
  function bits_to_natural(bits : in bit_vector) return natural;
  procedure int_to_bits(int : in integer; bits : out bit_vector);

  constant RR  : std_logic_vector(2 downto 0) := "001";
  constant RL  : std_logic_vector(2 downto 0) := "010";
  constant RI1 : std_logic_vector(2 downto 0) := "011";
  constant RI2 : std_logic_vector(2 downto 0) := "100";
  constant RK  : std_logic_vector(2 downto 0) := "101";
  constant RX  : std_logic_vector(2 downto 0) := "110";

  constant zero  : std_logic_vector(3 downto 0) := "0000";
  constant eight : std_logic_vector(3 downto 0) := "1000";
  constant A     : std_logic_vector(3 downto 0) := "1010";
  constant C     : std_logic_vector(3 downto 0) := "1100";
  constant E     : std_logic_vector(3 downto 0) := "1110";

  -- LOAD/STORE
     constant op_byte_load            : bit_8 := X"03" ;
     constant op_byte_load_index      : bit_8 := X"13" ;
     constant op_byte_store_index     : bit_8 := X"31" ;
     constant op_double_load_index    : bit_8 := X"1B" ;
     constant op_literal_load         : bit_8 := X"CC" ;
     constant op_load_multiple        : bit_8 := X"0F" ;
     constant op_load_mult_reg        : bit_8 := X"D2" ;
     constant op_store_mult_reg       : bit_8 := X"D6" ;
     constant op_word_load_index      : bit_8 := X"17" ;
     constant op_move_block           : bit_8 := X"2D" ;
     constant op_load                 : bit_8 := X"07" ;
     constant op_load_bit             : bit_8 := X"0A" ;
```



```
     constant op_load_double          : bit_8 := X"0B" ;
     constant op_load_mult            : bit_8 := X"00" ;
     constant op_byte_store           : bit_8 := X"23" ;
     constant op_store                : bit_8 := X"27" ;
     constant op_store_double         : bit_8 := X"2B" ;
     constant op_store_address        : bit_8 := X"B5" ;
     constant op_store_zero           : bit_8 := X"3F" ;
     constant op_decindex_store       : bit_8 := X"E5" ;
     constant op_decindex_store_double : bit_8 := X"E9";

  -- SUPPORT CHANNEL
     constant op_jump_io : bit_8 := X"0D" ;

  -- ARITHMETIC
     constant op_add              : bit_8 := X"4B";
     constant op_add_byte         : bit_8 := X"D7" ;
     constant op_add_double       : bit_8 := X"4F" ;
     constant op_subtract         : bit_8 := X"43" ;
     constant op_subtract_byte    : bit_8 := X"D3";
     constant op_subtract_double  : bit_8 := X"37";
     constant op_multiply         : bit_8 := X"5B";
     constant op_multiply_double  : bit_8 := X"BB" ;
     constant op_divide           : bit_8 := X"5F" ;
     constant op_divide_double    : bit_8 := X"BF";
     constant op_fpoint_add       : bit_8 := X"A7";
     constant op_fpoint_divide    : bit_8 := X"AF" ;
     constant op_fpoint_mult      : bit_8 := X"AB" ;
     constant op_fpoint_subtract  : bit_8 := X"A3";
     constant op_literal_add      : bit_8 := X"CA";
     constant op_literal_double   : bit_8 := X"CB";
     constant op_sign_ext_double  : bit_8 := X"E8";
     constant op_sign_ext         : bit_8 := X"E4";

  -- LOGICAL
     constant op_and : bit_8 := X"63" ;
     constant op_or  : bit_8 := X"67" ;
     constant op_xor : bit_8 := X"6B" ;

  -- COMPARE
     constant op_byte_comp    : bit_8 := X"DB";
     constant op_comp         : bit_8 := X"53" ;
     constant op_bit_comp     : bit_8 := X"1C" ;
     constant op_comp_double  : bit_8 := X"57" ;
     constant op_logical_comp : bit_8 := X"7B";
     constant op_masked_comp  : bit_8 := X"73" ;
     constant op_literal_comp : bit_8 := X"CD" ;


  -- JUMP
   --  constant op_jump : bit_8 := X"83" ;

  -- SHIFT
     constant op_alg_ldouble_shft     : bit_8 := X"3A" ;
     constant op_alg_left_shft        : bit_8 := X"32" ;
     constant op_alg_right_shft       : bit_8 := X"26";
     constant op_alg_rdouble_shft     : bit_8 := X"2E" ;
     constant op_cir_left_shft        : bit_8 := X"36" ;
     constant op_cir_ldouble_shft     : bit_8 := X"3E" ;
     constant op_literal_left_shft    : bit_8 := X"C5" ;
     constant op_literal_ldouble_shft : bit_8 := X"C7" ;
     constant op_literal_right_shft   : bit_8 := X"C0" ;
     constant op_literal_rdouble_shft : bit_8 := X"C2" ;
     constant op_logical_right_shft   : bit_8 := X"22" ;
```

```
    constant op_logical_rdouble_shft : bit_8 := X"2A" ;

  -- STACK and QUEUE
    constant op_stack_get_top    : bit_8 := X"1A" ;
    constant op_stack_put_top    : bit_8 := X"02" ;
    constant op_queue_get_top    : bit_8 := X"1E" ;
    constant op_queue_put_top    : bit_8 := X"12" ;
    constant op_queue_put_bottom : bit_8 := X"16" ;

  -- MISCELLANEOUS
    constant op_biased_fetch      : bit_8 := X"77" ;
    constant op_masked_substitute : bit_8 := X"6F" ;
    constant op_remote_execute    : bit_8 := X"76";
    constant op_set_bit           : bit_8 := X"14";
    constant op_zero_bit          : bit_8 := X"18";

  -- EXECUTIVE MODE
    constant op_load_addr_register : bit_8 := X"B3";
    constant op_load_physical_addr : bit_8 := X"E3";
    constant op_load_physical_location : bit_8 := X"EF";
    constant op_store_physical_location : bit_8 := X"E3";

-- ***** SPECIAL HANDLE *****
  -- LOAD/STORE
      constant op_load_store          : bit_8 := X"0C";
      constant op_store_monitor_clock : bit_8 := X"10";
  -- SUPPORT CHANNEL /JUMP
       constant op_jump                    : bit_8 := X"83" ;
       constant op_jump_input              : bit_8 := X"08";
       constant op_jump_link_memory        : bit_8 := X"8F";
       constant op_jump_link_register      : bit_8 := X"8B";
       constant op_jump_negative           : bit_8 := X"9F";
       constant op_jump_not_zero           : bit_8 := X"97";
       constant op_jump_positive           : bit_8 := X"9C";
       constant op_jump_zero               : bit_8 := X"93";
       constant op_local_jump_link_memory : bit_8 := X"8D";

  -- ARITHMETIC
       constant op_fixed_point : bit_8 := X"7E" ;
  -- COMPARE
      constant op_floating_comp : bit_8 := X"7C";


 -- Power PC Constant Define type

     constant op31 : bit_6 := "011111" ;
     constant op32 : bit_6 := "100000" ;
     constant op36 : bit_6 := "100100" ;

     constant ppc_0 : bit_5 := "00000" ;
     constant ppc_1 : bit_5 := "00001" ;
     constant ppc_31 : bit_5 := "11111";
     constant ppc_tmp : bit_5 := "01111";
     constant ppc_reg_default_content : std_logic_vector(31 downto 0) := "00000000000000000000000000000000

END Emulator_datatype;


PACKAGE body Emulator_datatype is



  function bits_to_int(bits : in bit_vector) return integer is
         variable temp : bit_vector(bits'range);
```



```
         variable result : integer := 0;
      begin
         if bits(bits'left)= '1' then
              temp := not bits ;
         else
            temp := bits ;
         end if;
             for index in bits'range loop
         --       for index in 0 to  bits'right loop
          --         result := result * 2 + bit'pos(temp(index));
                     result := result * 2 ;
                     if bits(index) = '1' then
                        result := result + 1 ;
                     end if ;
                end loop;
          if bits(bits'left) = '1' then
              result := (-result) - 1 ;
         end if ;
          return result ;
    end bits_to_int;

procedure int_to_bits(int : in integer; bits :out bit_vector) is
          variable temp : integer ;
          variable result : bit_vector(bits'range) ;
       begin
          if int < 0 then
             temp := -(int +1);
          else
             temp := int ;
          end if ;

      --    for index in bits'reverse_range loop
      --       result(index) := bit'val(temp rem 2);
           temp := temp/2 ;
    --       end loop;

          if int < 0 then
             result := not result;
             result(bits'left) := '1';
          end if;
          bits := result;
       end int_to_bits;

-- function resolve_bit_32(driver : in bit_32_array) return bit_32 is
--        constant floating_value : bit_32 := X"0000_0000";
--        variable result : bit_32 := floating_value;
--    begin
--        for i in driver'range loop
--           result := result or driver(i);
--        end loop;
--        return result;
--    end resolve_bit_32;

-- function resolve_bits_16(driver : in bits_16_array) return bits_16 is
--        constant floating_value : bits_16 := X"0000";
--        variable result : bits_16 := floating_value;
--    begin
--        for i in driver'range loop
--           result := result or driver(i);
--        end loop;
--        return result;
--    end resolve_bits_16;

 function bits_to_natural(bits: in bit_vector) return natural is
```

```
        variable result : natural := 0;
    begin
            for index in bits'range loop
                result := result * 2 ;
                if bits(index) = '1' then
                   result := result + 1 ;
                end if ;
            end loop;
    --     for index in bits'range loop
    --          result := result * 2 + bit'pos(bits(index));
    --     end loop;
         return result;
     end bits_to_natural;

END Emulator_datatype ;
```

```
-- Author:  George Phan
-- Date of netlist generation:  May-22-96
-- extended op (D) selection.
--------------------------------------------------------------------------

library IEEE;
-- library gsc1000d;
use IEEE.STD_LOGIC_1164.ALL;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.ALL;
--compass compile_off
use COMPASS_LIB.COMPASS_ETC.ALL;
--compass compile_on

entity em_sm   is

--compass compile_off
--compass compile_on


    port ( ayk_ins : in Std_Logic_Vector(15 downto 0) := (others => 'U');
           clk      : in  std_logic;
           sr1op    : in  std_logic;
           sr2op    : in  std_logic;
           inputdata: out std_logic_vector(15 downto 0);
           ydata: out std_logic_vector(15 downto 0);
           iw1data: out std_logic_vector(15 downto 0);
           iw2data: out std_logic_vector(15 downto 0);
           sr114  : out std_logic                     ;
           sr2hb  : out std_logic_vector(7  downto 0);
           edata  : out std_logic_vector(15 downto 0);
           rr       : out std_logic;
           ayk_dcyl : out std_logic;
           ayk_ecyl : out std_logic;
           ayk_w1cyl : out std_logic;
           ayk_w2cyl : out std_logic;
           ayk_w2dcyl : out std_logic;
           ayk_ycyl : out std_logic);
end em_sm;



--------------------------------------------------------------------------

architecture em_sm OF em_sm  is
--compass dp_gates;

--    component ni01d2
--        port ( I : in Std_Logic;

 --              Z : out Std_Logic );
 --   end component;

   CONSTANT delay : time := 0.8 ns;
```



```
signal  sm_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  sm_cycle1 : std_logic_vector(2 downto 0) := (others => 'U');
signal  ayk_idc   : std_logic_vector(1 downto 0) := (others => 'U');
signal  ayk_mdc   : std_logic_vector(3 downto 0) := (others => 'U');
signal  zz_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  zo_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  oz_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  m0oo_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  m1oo_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  ma0oo_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  ma1oo_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  maoo_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  moo_cycle0 : std_logic_vector(2 downto 0) := (others => 'U');
signal  zz_dcyl  : std_logic;
signal  zz_ecyl  : std_logic;
signal  zz_w1cyl  : std_logic;
signal  zz_w2cyl  : std_logic;
signal  zz_ycyl  : std_logic;
signal  zo_dcyl  : std_logic;
signal  zo_ecyl  : std_logic;
signal  zo_w1cyl  : std_logic;
signal  zo_w2cyl  : std_logic;
signal  zo_ycyl  : std_logic;
signal  oz_dcyl  : std_logic;
signal  oz_ecyl  : std_logic;
signal  oz_w1cyl  : std_logic;
signal  oz_w2cyl  : std_logic;
signal  oz_ycyl  : std_logic;
signal  m0oo_dcyl  : std_logic;
signal  m0oo_ecyl  : std_logic;
signal  m0oo_w1cyl  : std_logic;
signal  m0oo_w2cyl  : std_logic;
signal  m0oo_ycyl  : std_logic;
signal  m1oo_dcyl  : std_logic;
signal  m1oo_ecyl  : std_logic;
signal  m1oo_w1cyl  : std_logic;
signal  m1oo_w2cyl  : std_logic;
signal  m1oo_ycyl  : std_logic;
signal  ma0oo_dcyl  : std_logic;
signal  ma0oo_ecyl  : std_logic;
signal  ma0oo_w1cyl  : std_logic;
signal  ma0oo_w2cyl  : std_logic;
signal  ma0oo_ycyl  : std_logic;
signal  ma1oo_dcyl  : std_logic;
signal  ma1oo_ecyl  : std_logic;
signal  ma1oo_w1cyl  : std_logic;
signal  ma1oo_w2cyl  : std_logic;
signal  ma1oo_w2dcyl  : std_logic;
signal  ma1oo_ycyl  : std_logic;
signal  maoo_dcyl  : std_logic;
signal  maoo_ecyl  : std_logic;
signal  maoo_w1cyl  : std_logic;
signal  maoo_w2cyl  : std_logic;
signal  maoo_w2dcyl  : std_logic;
signal  maoo_ycyl  : std_logic;
signal  moo_dcyl  : std_logic;
signal  moo_ecyl  : std_logic;
signal  moo_w1cyl  : std_logic;
signal  moo_w2cyl  : std_logic;
signal  moo_w2dcyl  : std_logic;
signal  moo_ycyl  : std_logic;
signal  iw14      : std_logic;
signal  ayk_sr1cyl : std_logic;
signal  ayk_sr2cyl : std_logic;
```

```
signal  ydata0 : std_logic_vector(15 downto 0) := (others => 'U');
signal  inputdata0 : std_logic_vector(15 downto 0) := (others => 'U');
signal  iw1data0 : std_logic_vector(15 downto 0) := (others => 'U');
signal  iw2data0 : std_logic_vector(15 downto 0) := (others => 'U');
signal  sr1data  : std_logic_vector(15 downto 0) := (others => 'U');
signal  sr2data  : std_logic_vector(15 downto 0) := (others => 'U');
signal  sr1data0 : std_logic_vector(15 downto 0) := (others => 'U');
signal  sr2data0 : std_logic_vector(15 downto 0) := (others => 'U');
signal  edata0 : std_logic_vector(15 downto 0) := (others => 'U');
signal  ydata1 : std_logic_vector(15 downto 0) := (others => 'U');
signal  inputdata1 : std_logic_vector(15 downto 0) := (others => 'U');
signal  iw1data1 : std_logic_vector(15 downto 0) := (others => 'U');
signal  iw2data1 : std_logic_vector(15 downto 0) := (others => 'U');
signal  sr1data1 : std_logic_vector(15 downto 0) := (others => 'U');
signal  sr2data1 : std_logic_vector(15 downto 0) := (others => 'U');
signal  edata1 : std_logic_vector(15 downto 0) := (others => 'U');
signal      dcyl : std_logic;
signal      ecyl : std_logic;
signal      w1cyl : std_logic;
signal      w2cyl : std_logic;
signal      w2dcyl : std_logic;
signal      ycyl :  std_logic;
begin

   -- tielowninv: ni01d2
   --     port map ( I => vss, Z => tielow );


sm_idec:
process(ayk_ins)
begin
  ayk_idc <= ayk_ins(9) & ayk_ins(8) ;
end process sm_idec;

sm_mdec:
process(ayk_ins)
begin
  ayk_mdc <= ayk_ins(3) & ayk_ins(2) & ayk_ins(1) & ayk_ins(0);
end process sm_mdec;

sm_cntl:
process(sm_cycle1, ayk_ins,
   sm_cycle0 ,
   sm_cycle1 ,
   ayk_idc   ,
   ayk_mdc   ,
   zz_cycle0 ,
   zo_cycle0 ,
   oz_cycle0 ,
   m0oo_cycle0 ,
   m1oo_cycle0 ,
   ma0oo_cycle0 ,
   ma1oo_cycle0 ,
   maoo_cycle0 ,
   moo_cycle0 ,
   zz_dcyl  ,
   zz_ecyl  ,
   zz_w1cyl  ,
   zz_w2cyl  ,
   zz_ycyl  ,
   zo_dcyl  ,
   zo_ecyl  ,
   zo_w1cyl  ,
   zo_w2cyl  ,
```



```
   zo_ycyl  ,
   oz_dcyl  ,
   oz_ecyl  ,
   oz_w1cyl  ,
   oz_w2cyl  ,
   oz_ycyl  ,
   m0oo_dcyl ,
   m0oo_ecyl  ,
   m0oo_w1cyl ,
   m0oo_w2cyl ,
   m0oo_ycyl  ,
   m1oo_dcyl  ,
   m1oo_ecyl  ,
   m1oo_w1cyl  ,
   m1oo_w2cyl  ,
   m1oo_ycyl  ,
   ma0oo_dcyl  ,
   ma0oo_ecyl  ,
   ma0oo_w1cyl ,
   ma0oo_w2cyl ,
   ma0oo_ycyl  ,
   ma1oo_dcyl  ,
   ma1oo_ecyl  ,
   ma1oo_w1cyl ,
   ma1oo_w2cyl ,
   ma1oo_w2dcyl,
   ma1oo_ycyl  ,
   maoo_dcyl  ,
   maoo_ecyl  ,
   maoo_w1cyl ,
   maoo_w2cyl ,
   maoo_w2dcyl,
   maoo_ycyl  ,
   moo_dcyl  ,
   moo_ecyl  ,
   moo_w1cyl ,
   moo_w2cyl ,
   moo_w2dcyl,
   moo_ycyl  ,
   iw14  ,
    ayk_sr1cyl,
    ayk_sr2cyl,
    ydata0 ,
    inputdata0 ,
    iw1data0 ,
    iw2data0 ,
    sr1data  ,
    sr2data  ,
    sr1data0 ,
    sr2data0 ,
       dcyl,
       ecyl,
       w1cyl,
       w2cyl,
       w2dcyl,
       ycyl ,
    edata0 )
begin
      CASE sm_cycle1  is
      when "110"  => zz_cycle0 <= "000";
                     zz_dcyl <= '1';
                     zz_ecyl <= '0';
                     zz_w1cyl <= '0';
                     zz_w2cyl <= '0';
```

```
                zz_ycyl <= '0';
when others => zz_cycle0 <= "110" ;
                zz_dcyl <= '0';
                zz_ecyl <= '1';
                zz_w1cyl <= '0';
                zz_w2cyl <= '0';
                zz_ycyl <= '0';
END CASE;

CASE sm_cycle1  is
when "001" =>  zo_cycle0 <= "110";
                zo_dcyl <= '0';
                zo_ecyl <= '1';
                zo_w1cyl <= '0';
                zo_w2cyl <= '0';
                zo_ycyl <= '0';
when "110"  => zo_cycle0 <= "000";
                zo_dcyl <= '1';
                zo_ecyl <= '0';
                zo_w1cyl <= '0';
                zo_w2cyl <= '0';
                zo_ycyl <= '0';
when others => zo_cycle0 <= "001" ;
                zo_dcyl <= '1';
                zo_ecyl <= '0';
                zo_w1cyl <= '0';
                zo_w2cyl <= '0';
                zo_ycyl <= '0';
END CASE;

CASE sm_cycle1  is
when "001" =>  oz_cycle0 <= "110";
                oz_dcyl <= '0';
                oz_ecyl <= '1';
                oz_w1cyl <= '0';
                oz_w2cyl <= '0';
                oz_ycyl <= '0';
when "110"  => oz_cycle0 <= "000";
                oz_dcyl <= '1';
                oz_ecyl <= '0';
                oz_w1cyl <= '0';
                oz_w2cyl <= '0';
                oz_ycyl <= '0';
when others => oz_cycle0 <= "001" ;
                oz_dcyl <= '0';
                oz_ecyl <= '0';
                oz_w1cyl <= '0';
                oz_w2cyl <= '0';
                oz_ycyl <= '1';
END CASE;

   CASE sm_cycle1  is
   when "000" =>  m0oo_cycle0 <= "001";
                  m0oo_dcyl <= '0';
                  m0oo_ecyl <= '0';
                  m0oo_w1cyl <= '0';
                  m0oo_w2cyl <= '0';
                  m0oo_ycyl <= '1';
   when "001" =>  m0oo_cycle0 <= "110";
                  m0oo_dcyl <= '0';
                  m0oo_ecyl <= '1';
                  m0oo_w1cyl <= '0';
                  m0oo_w2cyl <= '0';
                  m0oo_ycyl <= '0';
```



```
when "110" =>  m0oo_cycle0 <= "000";
               m0oo_dcyl <= '1';
               m0oo_ecyl <= '0';
               m0oo_w1cyl <= '0';
               m0oo_w2cyl <= '0';
               m0oo_ycyl <= '0';
when others => m0oo_cycle0 <= "001";
               m0oo_dcyl <= '0';
               m0oo_ecyl <= '0';
               m0oo_w1cyl <= '0';
               m0oo_w2cyl <= '0';
               m0oo_ycyl <= '1';
END CASE;

 CASE sm_cycle1  is
 when "000" =>  ma0oo_cycle0 <= "001";
                ma0oo_dcyl <= '0';
                ma0oo_ecyl <= '0';
                ma0oo_w1cyl <= '0';
                ma0oo_w2cyl <= '0';
                ma0oo_ycyl <= '1';
 when "001" =>  ma0oo_cycle0 <= "010";
                ma0oo_dcyl <= '0';
                ma0oo_ecyl <= '0';
                ma0oo_w1cyl <= '1';
                ma0oo_w2cyl <= '0';
                ma0oo_ycyl <= '0';
 when "010" =>  ma0oo_cycle0 <= "011";
                ma0oo_dcyl <= '0';
                ma0oo_ecyl <= '0';
                ma0oo_w1cyl <= '0';
                ma0oo_w2cyl <= '1';
                ma0oo_ycyl <= '0';
 when "011" =>  ma0oo_cycle0 <= "110";
                ma0oo_dcyl <= '0';
                ma0oo_ecyl <= '1';
                ma0oo_w1cyl <= '0';
                ma0oo_w2cyl <= '0';
                ma0oo_ycyl <= '0';
 when "110" =>  ma0oo_cycle0 <= "000";
                ma0oo_dcyl <= '1';
                ma0oo_ecyl <= '0';
                ma0oo_w1cyl <= '0';
                ma0oo_w2cyl <= '0';
                ma0oo_ycyl <= '0';
 when others =>  ma0oo_cycle0 <= "001";
                ma0oo_dcyl <= '0';
                ma0oo_ecyl <= '0';
                ma0oo_w1cyl <= '0';
                ma0oo_w2cyl <= '0';
                ma0oo_ycyl <= '1';
  END CASE;

 CASE sm_cycle1  is
 when "000" =>  ma1oo_cycle0 <= "001";
                ma1oo_dcyl <= '0';
                ma1oo_ecyl <= '0';
                ma1oo_w1cyl <= '0';
                ma1oo_w2cyl <= '0';
                ma1oo_w2dcyl <= '0';
                ma1oo_ycyl <= '1';
 when "001" =>  ma1oo_cycle0 <= "010";
                ma1oo_dcyl <= '0';
                ma1oo_ecyl <= '0';
```

```
                maloo_w1cyl <= '1';
                maloo_w2cyl <= '0';
                maloo_w2dcyl <= '0';
                maloo_ycyl <= '0';
   when "010" => maloo_cycle0 <= "011";
                maloo_dcyl <= '0';
                maloo_ecyl <= '0';
                maloo_w1cyl <= '0';
                maloo_w2cyl <= '1';
                maloo_w2dcyl <= '0';
                maloo_ycyl <= '0';
   when "011" => maloo_cycle0 <= "101";
                maloo_dcyl <= '0';
                maloo_ecyl <= '0';
                maloo_w1cyl <= '0';
                maloo_w2cyl <= '0';
                maloo_w2dcyl <= '1';
                maloo_ycyl <= '0';
   when "101" => maloo_cycle0 <= "110";
                maloo_dcyl <= '0';
                maloo_ecyl <= '1';
                maloo_w1cyl <= '0';
                maloo_w2cyl <= '0';
                maloo_w2dcyl <= '0';
                maloo_ycyl <= '0';
   when "110" => maloo_cycle0 <= "000";
                maloo_dcyl <= '1';
                maloo_ecyl <= '0';
                maloo_w1cyl <= '0';
                maloo_w2cyl <= '0';
                maloo_w2dcyl <= '0';
                maloo_ycyl <= '0';
   when others => maloo_cycle0 <= "001";
                maloo_dcyl <= '0';
                maloo_ecyl <= '0';
                maloo_w1cyl <= '0';
                maloo_w2cyl <= '0';
                maloo_w2dcyl <= '0';
                maloo_ycyl <= '1';
    END CASE;


CASE sm_cycle1 is
when "001" => mloo_cycle0 <= "110";
              mloo_dcyl <= '0';
              mloo_ecyl <= '1';
              mloo_w1cyl <= '0';
              mloo_w2cyl <= '0';
              mloo_ycyl <= '0';
when "110"  => mloo_cycle0 <= "000";
              mloo_dcyl <= '1';
              mloo_ecyl <= '0';
              mloo_w1cyl <= '0';
              mloo_w2cyl <= '0';
              mloo_ycyl <= '0';
when others => mloo_cycle0 <= "001" ;
              mloo_dcyl <= '1';
              mloo_ecyl <= '0';
              mloo_w1cyl <= '0';
              mloo_w2cyl <= '0';
              mloo_ycyl <= '0';
END CASE;

CASE iw14 is
```



```
   when '0'    => maoo_cycle0 <= ma0oo_cycle0;
                maoo_dcyl <= ma0oo_dcyl;
                maoo_ecyl <= ma0oo_ecyl;
                maoo_w1cyl <= ma0oo_w1cyl;
                maoo_w2cyl <= ma0oo_w2cyl;
                maoo_w2dcyl <= '0';
                maoo_ycyl <= ma0oo_ycyl;
   when others => maoo_cycle0 <= maloo_cycle0;
                maoo_dcyl <= maloo_dcyl;
                maoo_ecyl <= maloo_ecyl;
                maoo_w1cyl <= maloo_w1cyl;
                maoo_w2cyl <= maloo_w2cyl;
                maoo_w2dcyl <= maloo_w2dcyl;
                maoo_ycyl <= maloo_ycyl;
END CASE;

CASE ayk_mdc is
 when "1000" =>  moo_cycle0 <= maoo_cycle0;
                moo_dcyl <= maoo_dcyl;
                moo_ecyl <= maoo_ecyl;
                moo_w1cyl <= maoo_w1cyl;
                moo_w2cyl <= maoo_w2cyl;
                moo_w2dcyl <= maoo_w2dcyl;
                moo_ycyl <= maoo_ycyl;
 when "1010" =>  moo_cycle0 <= maoo_cycle0;
                moo_dcyl <= maoo_dcyl;
                moo_ecyl <= maoo_ecyl;
                moo_w1cyl <= maoo_w1cyl;
                moo_w2cyl <= maoo_w2cyl;
                moo_w2dcyl <= maoo_w2dcyl;
                moo_ycyl <= maoo_ycyl;
 when "1100" =>  moo_cycle0 <= maoo_cycle0;
                moo_dcyl <= maoo_dcyl;
                moo_ecyl <= maoo_ecyl;
                moo_w1cyl <= maoo_w1cyl;
                moo_w2cyl <= maoo_w2cyl;
                moo_w2dcyl <= maoo_w2dcyl;
                moo_ycyl <= maoo_ycyl;
 when "1110" =>  moo_cycle0 <= maoo_cycle0;
                moo_dcyl <= maoo_dcyl;
                moo_ecyl <= maoo_ecyl;
                moo_w1cyl <= maoo_w1cyl;
                moo_w2cyl <= maoo_w2cyl;
                moo_w2dcyl <= maoo_w2dcyl;
                moo_ycyl <= maoo_ycyl;
 when "0000" =>  moo_cycle0 <= m0oo_cycle0;
                moo_dcyl <= m0oo_dcyl;
                moo_ecyl <= m0oo_ecyl;
                moo_w1cyl <= m0oo_w1cyl;
                moo_w2cyl <= m0oo_w2cyl;
                moo_w2dcyl <= '0';
                moo_ycyl <= m0oo_ycyl;
 when others  =>  moo_cycle0 <= mloo_cycle0;
                moo_dcyl <= mloo_dcyl;
                moo_ecyl <= mloo_ecyl;
                moo_w1cyl <= mloo_w1cyl;
                moo_w2cyl <= mloo_w2cyl;
                moo_w2dcyl <= '0';
                moo_ycyl <= mloo_ycyl;
END CASE;

Case ayk_idc is
   when "00"   =>  sm_cycle0 <= zz_cycle0;
                ayk_dcyl <= zz_dcyl;
```

```
                    ayk_ecyl <= zz_ecyl;
                    ayk_wlcyl <= zz_wlcyl;
                    ayk_w2cyl <= zz_w2cyl;
                    ayk_w2dcyl <= '0';
                    ayk_ycyl <= zz_ycyl;
                     dcyl <= zz_dcyl;
                     ecyl <= zz_ecyl;
                     wlcyl <= zz_wlcyl;
                     w2cyl <= zz_w2cyl;
                     w2dcyl <= '0';
                     ycyl <= zz_ycyl;
                     RR   <= '1';
      when "01"    =>  sm_cycle0 <= zo_cycle0;
                    ayk_dcyl <= zo_dcyl;
                    ayk_ecyl <= zo_ecyl;
                    ayk_wlcyl <= zo_wlcyl;
                    ayk_w2cyl <= zo_w2cyl;
                    ayk_w2dcyl <= '0';
                    ayk_ycyl <= zo_ycyl;
                     dcyl <= zo_dcyl;
                     ecyl <= zo_ecyl;
                     wlcyl <= zo_wlcyl;
                     w2cyl <= zo_w2cyl;
                     w2dcyl <= '0';
                     ycyl <= zo_ycyl;
      when "10"    =>  sm_cycle0 <= oz_cycle0;
                    ayk_dcyl <= oz_dcyl;
                    ayk_ecyl <= oz_ecyl;
                    ayk_wlcyl <= oz_wlcyl;
                    ayk_w2cyl <= oz_w2cyl;
                    ayk_w2dcyl <= '0';
                    ayk_ycyl <= oz_ycyl;
                     dcyl <= oz_dcyl;
                     ecyl <= oz_ecyl;
                     wlcyl <= oz_wlcyl;
                     w2cyl <= oz_w2cyl;
                     w2dcyl <= '0';
                     ycyl <= oz_ycyl;
      when others =>  sm_cycle0 <= moo_cycle0;
                    ayk_dcyl <= moo_dcyl;
                    ayk_ecyl <= moo_ecyl;
                    ayk_wlcyl <= moo_wlcyl;
                    ayk_w2cyl <= moo_w2cyl;
                    ayk_w2dcyl <= moo_w2dcyl;
                    ayk_ycyl <= moo_ycyl;
                     dcyl <= moo_dcyl;
                     ecyl <= moo_ecyl;
                     wlcyl <= moo_wlcyl;
                     w2cyl <= moo_w2cyl;
                     w2dcyl <= moo_w2dcyl;
                     ycyl <= moo_ycyl;
                     RR   <= '0';
     END CASE;

   END process sm_cntl;

reg2:
process(clk, sm_cycle0)
begin

  If (rising_edge(clk)) THEN
       sm_cycle1 <= sm_cycle0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
```



```
      sm_cycle1 <= "XXX";
--compass compile_on
    END IF;
  END PROCESS reg2;

ymux:
process(ycyl, ydata1, ayk_ins)
begin
case ycyl is
 when '1'  =>
   ydata0 <= ydata1 ;
  when others  =>
   ydata0 <= ayk_ins;
 END CASE;
END PROCESS ymux;

yreg:
process(clk, ydata0 )
begin

  If (rising_edge(clk)) THEN
       ydata1 <= ydata0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
         ydata1 <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS yreg;

ybuf:
process(ydata1)
begin
  ydata <= ydata1 ;
end process ybuf;

inputbuf:
process(inputdata1)
begin
  inputdata <= inputdata1 ;
end process inputbuf;

inputmux:
process(dcyl, inputdata1, ayk_ins)
begin
case dcyl is
  when '1'  =>
   inputdata0 <= inputdata1 ;
  when others  =>
   inputdata0 <= ayk_ins;
 END CASE;
END PROCESS inputmux;

inputreg:
process(clk, inputdata0 )
begin

  If (rising_edge(clk)) THEN
       inputdata1 <= inputdata0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
         inputdata1 <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS inputreg;
```

```
iw14gen:
process(iw1data1)
begin
   iw14 <= iw1data1(14) ;
END PROCESS iw14gen;

iw1buf:
process(iw1data1)
begin
  iw1data <= iw1data1 ;
end process iw1buf;

iw1mux:
process(w1cyl, iw1data1, ayk_ins)
begin
case w1cyl is
  when '1'  =>
   iw1data0 <= iw1data1 ;
  when others  =>
   iw1data0 <= ayk_ins;
 END CASE;
END PROCESS iw1mux;

iw1reg:
process(clk, iw1data0 )
begin

  If (rising_edge(clk)) THEN
       iw1data1 <= iw1data0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
         iw1data1 <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS iw1reg;

iw2buf:
process(iw2data1)
begin
  iw2data <= iw2data1 ;
end process iw2buf;

iw2mux:
process(w2cyl, iw2data1, ayk_ins)
begin
case w2cyl is
  when '1'  =>
   iw2data0 <= iw2data1 ;
  when others  =>
   iw2data0 <= ayk_ins;
 END CASE;
END PROCESS iw2mux;

iw2reg:
process(clk, iw2data0 )
begin

  If (rising_edge(clk)) THEN
       iw2data1 <= iw2data0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
         iw2data1 <= "XXXXXXXXXXXXXXXX";
--compass compile_on
```



```
    END IF;
  END PROCESS iw2reg;

sr2buf:
process(sr2data1)
begin
  sr2data <= sr2data1 ;
  sr2hb   <= sr2data1(15 downto 8) ;
end process sr2buf;

sr2gen:
process(ecy1, sr2op)
begin
  ayk_sr2cyl <= ecyl and sr2op ;
END PROCESS sr2gen;

sr2mux:
process(ayk_sr2cyl, sr2data1, ayk_ins)
begin
case ayk_sr2cyl is
  when '1'  =>
   sr2data0 <= sr2data1 ;
  when others  =>
   sr2data0 <= ayk_ins;
 END case;
END PROCESS sr2mux;

sr2reg:
process(clk, sr2data0 )
begin

  If (rising_edge(clk)) THEN
       sr2data1 <= sr2data0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
         sr2data1 <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS sr2reg;

sr1gen:
process(ecyl, sr1op)
begin
  ayk_sr1cyl <= ecyl and sr1op ;
END PROCESS sr1gen;

sr1buf:
process(sr1data1)
begin
  sr1data <= sr1data1 ;
  sr114   <= sr1data1(14) ;
end process sr1buf;

sr1mux:
process(ayk_sr1cyl, sr1data1, ayk_ins)
begin
case ayk_sr1cyl is
  when '1'  =>
   sr1data0 <= sr1data1 ;
  when others  =>
   sr1data0 <= ayk_ins;
 END CASE;
END PROCESS sr1mux;
```

```
sr1reg:
process(clk, sr1data0 )
begin

  If (rising_edge(clk)) THEN
       sr1data1 <= sr1data0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
         sr1data1 <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS sr1reg;

ebuf:
process(edata1)
begin
  edata <= edata1 ;
end process ebuf;

emux:
process(ecy1, edata1, ayk_ins)
begin
case ecy1 is
  when '1'  =>
   edata0 <= edata1 ;
  when others  =>
   edata0 <= ayk_ins;
 END CASE;
END PROCESS emux;

ereg:
process(clk, edata0 )
begin

  If (rising_edge(clk)) THEN
       edata1 <= edata0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
         edata1 <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS ereg;



end em_sm;
--compass compile_off



configuration em_sm_CON of em_sm is
  for em_sm
  end for;
end em_sm_CON;
--compass compile_on
-------------------------------------------------
```

```
-- Author:  George Phan
-- Date of netlist generation:  MAY-30-96
-- controller for emulator process.
--------------------------------------------------------------------------

library IEEE;
-- library gsc1000d;
use IEEE.STD_LOGIC_1164.ALL;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.ALL;
--compass compile_off
use COMPASS_LIB.COMPASS_ETC.ALL;
--compass compile_on

entity em_intf is

--compass compile_off
--compass compile_on


    port ( rom_op : in Std_Logic_Vector(16 downto 0) := (others => 'U');
           ppc_addr  : in  Std_Logic_Vector(4 downto 0) := (others => 'U');
           new_ppc_ins : in  Std_Logic;
           wppc_addr_dec : out Std_Logic_Vector(31 downto 0) := (others => 'U');
           rppc_addr_dec : out Std_Logic_Vector(31 downto 0) := (others => 'U');
           wd_addr_dec : out Std_Logic_Vector(31 downto 0) := (others => 'U');
           wpcir_addr_dec : out Std_Logic_Vector(31 downto 0) := (others => 'U');
           CLK : in Std_Logic := 'U');
end em_intf;



--------------------------------------------------------------------------

architecture em_intf OF em_intf is
--compass dp_gates;

signal next_pcir_addr : std_logic_vector (4 downto 0);
signal wpcir_addr : std_logic_vector (4 downto 0);
signal pcir_addr : std_logic_vector (4 downto 0);
signal wppc_addr : std_logic_vector (4 downto 0);
signal rppc_addr : std_logic_vector (4 downto 0);
signal wd_addr : std_logic_vector (4 downto 0);

--    component ni01d2
--        port ( I : in Std_Logic;

 --             Z : out Std_Logic );
 --   end component;

   CONSTANT delay : time := 0.8 ns;



begin

   -- tielowninv: ni01d2
   --      port map ( I => vss, Z => tielow );


reg1:
```



```
process(clk, next_pcir_addr,
       wpcir_addr ,
       wppc_addr ,
       rppc_addr ,
       wd_addr )

begin
  If (rising_edge(clk)) THEN
     pcir_addr <= next_pcir_addr(4 downto 0) after delay;

--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
     pcir_addr <= "XXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS reg1;

  pcir_addr_sel:
  PROCESS(pcir_addr,new_ppc_ins)
  BEGIN
   case new_ppc_ins is
     when '1'  => wpcir_addr <= "00000" ;
     when others  => wpcir_addr <= pcir_addr ;
   end case ;
  END PROCESS pcir_addr_sel;

  picr_wr_sel:
  PROCESS(rom_op,ppc_addr)
  BEGIN
    case rom_op(16 downto 11) is
      when "101010" => wd_addr <= rom_op(4 downto 0) ;
      when "100100" => rppc_addr <= ppc_addr ;
      when others   => rppc_addr <= "00000" ;
                       wd_addr  <= "00000" ;
                       wppc_addr  <= ppc_addr;
    end case;

   end process picr_wr_sel;

  pcir_addr_dec:
  PROCESS( wpcir_addr)
  BEGIN
      next_pcir_addr <=  wpcir_addr + '1';
     case wpcir_addr is
        when "00000"  =>
          wpcir_addr_dec(0) <= '1' ;
        when others   =>
          wpcir_addr_dec(0) <= '0' ;
      end case;
     case wpcir_addr is
        when "00001"  =>
          wpcir_addr_dec(1) <= '1' ;
        when others   =>
          wpcir_addr_dec(1) <= '0' ;
      end case;
     case wpcir_addr is
        when "00010"  =>
          wpcir_addr_dec(2) <= '1' ;
        when others   =>
          wpcir_addr_dec(2) <= '0' ;
      end case;
     case wpcir_addr is
        when "00011"  =>
          wpcir_addr_dec(3) <= '1' ;
```

```
    when others   =>
      wpcir_addr_dec(3) <= '0' ;
 end case;
case wpcir_addr is
   when "00100"  =>
     wpcir_addr_dec(4) <= '1' ;
   when others   =>
     wpcir_addr_dec(4) <= '0' ;
 end case;
case wpcir_addr is
   when "00101"  =>
     wpcir_addr_dec(5) <= '1' ;
   when others   =>
     wpcir_addr_dec(5) <= '0' ;
 end case;
case wpcir_addr is
   when "00110"  =>
     wpcir_addr_dec(6) <= '1' ;
   when others   =>
     wpcir_addr_dec(6) <= '0' ;
 end case;
case wpcir_addr is
   when "00111"  =>
     wpcir_addr_dec(7) <= '1' ;
   when others   =>
     wpcir_addr_dec(7) <= '0' ;
 end case;
case wpcir_addr is
   when "01000"  =>
     wpcir_addr_dec(8) <= '1' ;
   when others   =>
     wpcir_addr_dec(8) <= '0' ;
 end case;
case wpcir_addr is
   when "01001"  =>
     wpcir_addr_dec(9) <= '1' ;
   when others   =>
     wpcir_addr_dec(9) <= '0' ;
 end case;
case wpcir_addr is
   when "01010"  =>
     wpcir_addr_dec(10) <= '1' ;
   when others   =>
     wpcir_addr_dec(10) <= '0' ;
 end case;
case wpcir_addr is
   when "01011"  =>
     wpcir_addr_dec(11) <= '1' ;
   when others   =>
     wpcir_addr_dec(11) <= '0' ;
 end case;
case wpcir_addr is
   when "01100"  =>
     wpcir_addr_dec(12) <= '1' ;
   when others   =>
     wpcir_addr_dec(12) <= '0' ;
 end case;
case wpcir_addr is
   when "01101"  =>
     wpcir_addr_dec(13) <= '1' ;
   when others   =>
     wpcir_addr_dec(13) <= '0' ;
 end case;
case wpcir_addr is
```



```
   when "01110"  =>
     wpcir_addr_dec(14) <= '1' ;
   when others   =>
     wpcir_addr_dec(14) <= '0' ;
 end case;
case wpcir_addr is
   when "01111"  =>
     wpcir_addr_dec(15) <= '1' ;
   when others   =>
     wpcir_addr_dec(15) <= '0' ;
 end case;
case wpcir_addr is
   when "10000"  =>
     wpcir_addr_dec(16) <= '1' ;
   when others   =>
     wpcir_addr_dec(16) <= '0' ;
 end case;
case wpcir_addr is
   when "10001"  =>
     wpcir_addr_dec(17) <= '1' ;
   when others   =>
     wpcir_addr_dec(17) <= '0' ;
 end case;
case wpcir_addr is
   when "10010"  =>
     wpcir_addr_dec(18) <= '1' ;
   when others   =>
     wpcir_addr_dec(18) <= '0' ;
 end case;
case wpcir_addr is
   when "10011"  =>
     wpcir_addr_dec(19) <= '1' ;
   when others   =>
     wpcir_addr_dec(19) <= '0' ;
 end case;
case wpcir_addr is
   when "10100"  =>
     wpcir_addr_dec(20) <= '1' ;
   when others   =>
     wpcir_addr_dec(20) <= '0' ;
 end case;
case wpcir_addr is
   when "10101"  =>
     wpcir_addr_dec(21) <= '1' ;
   when others   =>
     wpcir_addr_dec(21) <= '0' ;
 end case;
case wpcir_addr is
   when "10110"  =>
     wpcir_addr_dec(22) <= '1' ;
   when others   =>
     wpcir_addr_dec(22) <= '0' ;
 end case;
case wpcir_addr is
   when "10111"  =>
     wpcir_addr_dec(23) <= '1' ;
   when others   =>
     wpcir_addr_dec(23) <= '0' ;
 end case;
case wpcir_addr is
   when "11000"  =>
     wpcir_addr_dec(24) <= '1' ;
   when others   =>
     wpcir_addr_dec(24) <= '0' ;
```

```
    end case;
  case wpcir_addr is
     when "11001"  =>
       wpcir_addr_dec(25) <= '1' ;
     when others   =>
       wpcir_addr_dec(25) <= '0' ;
    end case;
  case wpcir_addr is
     when "11010"  =>
       wpcir_addr_dec(26) <= '1' ;
     when others   =>
       wpcir_addr_dec(26) <= '0' ;
    end case;
  case wpcir_addr is
     when "11011"  =>
       wpcir_addr_dec(27) <= '1' ;
     when others   =>
       wpcir_addr_dec(27) <= '0' ;
    end case;
  case wpcir_addr is
     when "11100"  =>
       wpcir_addr_dec(28) <= '1' ;
     when others   =>
       wpcir_addr_dec(28) <= '0' ;
    end case;
  case wpcir_addr is
     when "11101"  =>
       wpcir_addr_dec(29) <= '1' ;
     when others   =>
       wpcir_addr_dec(29) <= '0' ;
    end case;
  case wpcir_addr is
     when "11110"  =>
       wpcir_addr_dec(30) <= '1' ;
     when others   =>
       wpcir_addr_dec(30) <= '0' ;
    end case;
  case wpcir_addr is
     when "11111"  =>
       wpcir_addr_dec(31) <= '1' ;
     when others   =>
       wpcir_addr_dec(31) <= '0' ;
    end case;
END PROCESS pcir_addr_dec;

wppc_dec:
PROCESS( wppc_addr)
begin
   case wppc_addr is
      when "00000"  =>
        wppc_addr_dec(0) <= '1' ;
      when others   =>
        wppc_addr_dec(0) <= '0' ;
    end case;
   case wppc_addr is
      when "00001"  =>
        wppc_addr_dec(1) <= '1' ;
      when others   =>
        wppc_addr_dec(1) <= '0' ;
    end case;
   case wppc_addr is
      when "00010"  =>
        wppc_addr_dec(2) <= '1' ;
      when others   =>
```



```
        wppc_addr_dec(2) <= '0' ;
    end case;
  case wppc_addr is
     when "00011"  =>
       wppc_addr_dec(3) <= '1' ;
     when others   =>
       wppc_addr_dec(3) <= '0' ;
    end case;
  case wppc_addr is
     when "00100"  =>
       wppc_addr_dec(4) <= '1' ;
     when others   =>
       wppc_addr_dec(4) <= '0' ;
    end case;
  case wppc_addr is
     when "00101"  =>
       wppc_addr_dec(5) <= '1' ;
     when others   =>
       wppc_addr_dec(5) <= '0' ;
    end case;
  case wppc_addr is
     when "00110"  =>
       wppc_addr_dec(6) <= '1' ;
     when others   =>
       wppc_addr_dec(6) <= '0' ;
    end case;
  case wppc_addr is
     when "00111"  =>
       wppc_addr_dec(7) <= '1' ;
     when others   =>
       wppc_addr_dec(7) <= '0' ;
    end case;
  case wppc_addr is
     when "01000"  =>
       wppc_addr_dec(8) <= '1' ;
     when others   =>
       wppc_addr_dec(8) <= '0' ;
    end case;
  case wppc_addr is
     when "01001"  =>
       wppc_addr_dec(9) <= '1' ;
     when others   =>
       wppc_addr_dec(9) <= '0' ;
    end case;
  case wppc_addr is
     when "01010"  =>
       wppc_addr_dec(10) <= '1' ;
     when others   =>
       wppc_addr_dec(10) <= '0' ;
    end case;
  case wppc_addr is
     when "01011"  =>
       wppc_addr_dec(11) <= '1' ;
     when others   =>
       wppc_addr_dec(11) <= '0' ;
    end case;
  case wppc_addr is
     when "01100"  =>
       wppc_addr_dec(12) <= '1' ;
     when others   =>
       wppc_addr_dec(12) <= '0' ;
    end case;
  case wppc_addr is
     when "01101"  =>
```

```
      wppc_addr_dec(13) <= '1' ;
    when others   =>
      wppc_addr_dec(13) <= '0' ;
 end case;
case wppc_addr is
   when "01110"  =>
     wppc_addr_dec(14) <= '1' ;
   when others   =>
     wppc_addr_dec(14) <= '0' ;
 end case;
case wppc_addr is
   when "01111"  =>
     wppc_addr_dec(15) <= '1' ;
   when others   =>
     wppc_addr_dec(15) <= '0' ;
 end case;
case wppc_addr is
   when "10000"  =>
     wppc_addr_dec(16) <= '1' ;
   when others   =>
     wppc_addr_dec(16) <= '0' ;
 end case;
case wppc_addr is
   when "10001"  =>
     wppc_addr_dec(17) <= '1' ;
   when others   =>
     wppc_addr_dec(17) <= '0' ;
 end case;
case wppc_addr is
   when "10010"  =>
     wppc_addr_dec(18) <= '1' ;
   when others   =>
     wppc_addr_dec(18) <= '0' ;
 end case;
case wppc_addr is
   when "10011"  =>
     wppc_addr_dec(19) <= '1' ;
   when others   =>
     wppc_addr_dec(19) <= '0' ;
 end case;
case wppc_addr is
   when "10100"  =>
     wppc_addr_dec(20) <= '1' ;
   when others   =>
     wppc_addr_dec(20) <= '0' ;
 end case;
case wppc_addr is
   when "10101"  =>
     wppc_addr_dec(21) <= '1' ;
   when others   =>
     wppc_addr_dec(21) <= '0' ;
 end case;
case wppc_addr is
   when "10110"  =>
     wppc_addr_dec(22) <= '1' ;
   when others   =>
     wppc_addr_dec(22) <= '0' ;
 end case;
case wppc_addr is
   when "10111"  =>
     wppc_addr_dec(23) <= '1' ;
   when others   =>
     wppc_addr_dec(23) <= '0' ;
 end case;
```



```
case wppc_addr is
   when "11000"  =>
     wppc_addr_dec(24) <= '1' ;
   when others   =>
     wppc_addr_dec(24) <= '0' ;
 end case;
case wppc_addr is
   when "11001"  =>
     wppc_addr_dec(25) <= '1' ;
   when others   =>
     wppc_addr_dec(25) <= '0' ;
 end case;
case wppc_addr is
   when "11010"  =>
     wppc_addr_dec(26) <= '1' ;
   when others   =>
     wppc_addr_dec(26) <= '0' ;
 end case;
case wppc_addr is
   when "11011"  =>
     wppc_addr_dec(27) <= '1' ;
   when others   =>
     wppc_addr_dec(27) <= '0' ;
 end case;
case wppc_addr is
   when "11100"  =>
     wppc_addr_dec(28) <= '1' ;
   when others   =>
     wppc_addr_dec(28) <= '0' ;
 end case;
case wppc_addr is
   when "11101"  =>
     wppc_addr_dec(29) <= '1' ;
   when others   =>
     wppc_addr_dec(29) <= '0' ;
 end case;
case wppc_addr is
   when "11110"  =>
     wppc_addr_dec(30) <= '1' ;
   when others   =>
     wppc_addr_dec(30) <= '0' ;
 end case;
case wppc_addr is
   when "11111"  =>
     wppc_addr_dec(31) <= '1' ;
   when others   =>
     wppc_addr_dec(31) <= '0' ;
 end case;
end process wppc_dec;

rppc_dec:
PROCESS( rppc_addr)
begin
   case rppc_addr is
      when "00000"  =>
        rppc_addr_dec(0) <= '1' ;
      when others   =>
        rppc_addr_dec(0) <= '0' ;
    end case;
   case rppc_addr is
      when "00001"  =>
        rppc_addr_dec(1) <= '1' ;
      when others   =>
        rppc_addr_dec(1) <= '0' ;
```

```
 end case;
case rppc_addr is
   when "00010"  =>
     rppc_addr_dec(2) <= '1' ;
   when others   =>
     rppc_addr_dec(2) <= '0' ;
 end case;
case rppc_addr is
   when "00011"  =>
     rppc_addr_dec(3) <= '1' ;
   when others   =>
     rppc_addr_dec(3) <= '0' ;
 end case;
case rppc_addr is
   when "00100"  =>
     rppc_addr_dec(4) <= '1' ;
   when others   =>
     rppc_addr_dec(4) <= '0' ;
 end case;
case rppc_addr is
   when "00101"  =>
     rppc_addr_dec(5) <= '1' ;
   when others   =>
     rppc_addr_dec(5) <= '0' ;
 end case;
case rppc_addr is
   when "00110"  =>
     rppc_addr_dec(6) <= '1' ;
   when others   =>
     rppc_addr_dec(6) <= '0' ;
 end case;
case rppc_addr is
   when "00111"  =>
     rppc_addr_dec(7) <= '1' ;
   when others   =>
     rppc_addr_dec(7) <= '0' ;
 end case;
case rppc_addr is
   when "01000"  =>
     rppc_addr_dec(8) <= '1' ;
   when others   =>
     rppc_addr_dec(8) <= '0' ;
 end case;
case rppc_addr is
   when "01001"  =>
     rppc_addr_dec(9) <= '1' ;
   when others   =>
     rppc_addr_dec(9) <= '0' ;
 end case;
case rppc_addr is
   when "01010"  =>
     rppc_addr_dec(10) <= '1' ;
   when others   =>
     rppc_addr_dec(10) <= '0' ;
 end case;
case rppc_addr is
   when "01011"  =>
     rppc_addr_dec(11) <= '1' ;
   when others   =>
     rppc_addr_dec(11) <= '0' ;
 end case;
case rppc_addr is
   when "01100"  =>
     rppc_addr_dec(12) <= '1' ;
```



```
   when others   =>
     rppc_addr_dec(12) <= '0' ;
 end case;
case rppc_addr is
   when "01101"  =>
     rppc_addr_dec(13) <= '1' ;
   when others   =>
     rppc_addr_dec(13) <= '0' ;
 end case;
case rppc_addr is
   when "01110"  =>
     rppc_addr_dec(14) <= '1' ;
   when others   =>
     rppc_addr_dec(14) <= '0' ;
 end case;
case rppc_addr is
   when "01111"  =>
     rppc_addr_dec(15) <= '1' ;
   when others   =>
     rppc_addr_dec(15) <= '0' ;
 end case;
case rppc_addr is
   when "10000"  =>
     rppc_addr_dec(16) <= '1' ;
   when others   =>
     rppc_addr_dec(16) <= '0' ;
 end case;
case rppc_addr is
   when "10001"  =>
     rppc_addr_dec(17) <= '1' ;
   when others   =>
     rppc_addr_dec(17) <= '0' ;
 end case;
case rppc_addr is
   when "10010"  =>
     rppc_addr_dec(18) <= '1' ;
   when others   =>
     rppc_addr_dec(18) <= '0' ;
 end case;
case rppc_addr is
   when "10011"  =>
     rppc_addr_dec(19) <= '1' ;
   when others   =>
     rppc_addr_dec(19) <= '0' ;
 end case;
case rppc_addr is
   when "10100"  =>
     rppc_addr_dec(20) <= '1' ;
   when others   =>
     rppc_addr_dec(20) <= '0' ;
 end case;
case rppc_addr is
   when "10101"  =>
     rppc_addr_dec(21) <= '1' ;
   when others   =>
     rppc_addr_dec(21) <= '0' ;
 end case;
case rppc_addr is
   when "10110"  =>
     rppc_addr_dec(22) <= '1' ;
   when others   =>
     rppc_addr_dec(22) <= '0' ;
 end case;
case rppc_addr is
```

```
      when "10111"  =>
        rppc_addr_dec(23) <= '1' ;
      when others   =>
        rppc_addr_dec(23) <= '0' ;
    end case;
   case rppc_addr is
      when "11000"  =>
        rppc_addr_dec(24) <= '1' ;
      when others   =>
        rppc_addr_dec(24) <= '0' ;
    end case;
   case rppc_addr is
      when "11001"  =>
        rppc_addr_dec(25) <= '1' ;
      when others   =>
        rppc_addr_dec(25) <= '0' ;
    end case;
   case rppc_addr is
      when "11010"  =>
        rppc_addr_dec(26) <= '1' ;
      when others   =>
        rppc_addr_dec(26) <= '0' ;
    end case;
   case rppc_addr is
      when "11011"  =>
        rppc_addr_dec(27) <= '1' ;
      when others   =>
        rppc_addr_dec(27) <= '0' ;
    end case;
   case rppc_addr is
      when "11100"  =>
        rppc_addr_dec(28) <= '1' ;
      when others   =>
        rppc_addr_dec(28) <= '0' ;
    end case;
   case rppc_addr is
      when "11101"  =>
        rppc_addr_dec(29) <= '1' ;
      when others   =>
        rppc_addr_dec(29) <= '0' ;
    end case;
   case rppc_addr is
      when "11110"  =>
        rppc_addr_dec(30) <= '1' ;
      when others   =>
        rppc_addr_dec(30) <= '0' ;
    end case;
   case rppc_addr is
      when "11111"  =>
        rppc_addr_dec(31) <= '1' ;
      when others   =>
        rppc_addr_dec(31) <= '0' ;
    end case;
end process rppc_dec;

wd_dec:
PROCESS( wd_addr)
begin
   case wd_addr is
      when "00000"  =>
        wd_addr_dec(0) <= '1' ;
      when others   =>
        wd_addr_dec(0) <= '0' ;
    end case;
```



```
case wd_addr is
   when "00001"  =>
     wd_addr_dec(1) <= '1' ;
   when others   =>
     wd_addr_dec(1) <= '0' ;
 end case;
case wd_addr is
   when "00010"  =>
     wd_addr_dec(2) <= '1' ;
   when others   =>
     wd_addr_dec(2) <= '0' ;
 end case;
case wd_addr is
   when "00011"  =>
     wd_addr_dec(3) <= '1' ;
   when others   =>
     wd_addr_dec(3) <= '0' ;
 end case;
case wd_addr is
   when "00100"  =>
     wd_addr_dec(4) <= '1' ;
   when others   =>
     wd_addr_dec(4) <= '0' ;
 end case;
case wd_addr is
   when "00101"  =>
     wd_addr_dec(5) <= '1' ;
   when others   =>
     wd_addr_dec(5) <= '0' ;
 end case;
case wd_addr is
   when "00110"  =>
     wd_addr_dec(6) <= '1' ;
   when others   =>
     wd_addr_dec(6) <= '0' ;
 end case;
case wd_addr is
   when "00111"  =>
     wd_addr_dec(7) <= '1' ;
   when others   =>
     wd_addr_dec(7) <= '0' ;
 end case;
case wd_addr is
   when "01000"  =>
     wd_addr_dec(8) <= '1' ;
   when others   =>
     wd_addr_dec(8) <= '0' ;
 end case;
case wd_addr is
   when "01001"  =>
     wd_addr_dec(9) <= '1' ;
   when others   =>
     wd_addr_dec(9) <= '0' ;
 end case;
case wd_addr is
   when "01010"  =>
     wd_addr_dec(10) <= '1' ;
   when others   =>
     wd_addr_dec(10) <= '0' ;
 end case;
case wd_addr is
   when "01011"  =>
     wd_addr_dec(11) <= '1' ;
   when others   =>
```

```
        wd_addr_dec(11) <= '0' ;
    end case;
    case wd_addr is
      when "01100"  =>
        wd_addr_dec(12) <= '1' ;
      when others   =>
        wd_addr_dec(12) <= '0' ;
    end case;
    case wd_addr is
      when "01101"  =>
        wd_addr_dec(13) <= '1' ;
      when others   =>
        wd_addr_dec(13) <= '0' ;
    end case;
    case wd_addr is
      when "01110"  =>
        wd_addr_dec(14) <= '1' ;
      when others   =>
        wd_addr_dec(14) <= '0' ;
    end case;
    case wd_addr is
      when "01111"  =>
        wd_addr_dec(15) <= '1' ;
      when others   =>
        wd_addr_dec(15) <= '0' ;
    end case;
    case wd_addr is
      when "10000"  =>
        wd_addr_dec(16) <= '1' ;
      when others   =>
        wd_addr_dec(16) <= '0' ;
    end case;
    case wd_addr is
      when "10001"  =>
        wd_addr_dec(17) <= '1' ;
      when others   =>
        wd_addr_dec(17) <= '0' ;
    end case;
    case wd_addr is
      when "10010"  =>
        wd_addr_dec(18) <= '1' ;
      when others   =>
        wd_addr_dec(18) <= '0' ;
    end case;
    case wd_addr is
      when "10011"  =>
        wd_addr_dec(19) <= '1' ;
      when others   =>
        wd_addr_dec(19) <= '0' ;
    end case;
    case wd_addr is
      when "10100"  =>
        wd_addr_dec(20) <= '1' ;
      when others   =>
        wd_addr_dec(20) <= '0' ;
    end case;
    case wd_addr is
      when "10101"  =>
        wd_addr_dec(21) <= '1' ;
      when others   =>
        wd_addr_dec(21) <= '0' ;
    end case;
    case wd_addr is
      when "10110"  =>
```



```
        wd_addr_dec(22) <= '1' ;
      when others   =>
        wd_addr_dec(22) <= '0' ;
    end case;
    case wd_addr is
      when "10111"  =>
        wd_addr_dec(23) <= '1' ;
      when others   =>
        wd_addr_dec(23) <= '0' ;
    end case;
    case wd_addr is
      when "11000"  =>
        wd_addr_dec(24) <= '1' ;
      when others   =>
        wd_addr_dec(24) <= '0' ;
    end case;
    case wd_addr is
      when "11001"  =>
        wd_addr_dec(25) <= '1' ;
      when others   =>
        wd_addr_dec(25) <= '0' ;
    end case;
    case wd_addr is
      when "11010"  =>
        wd_addr_dec(26) <= '1' ;
      when others   =>
        wd_addr_dec(26) <= '0' ;
    end case;
    case wd_addr is
      when "11011"  =>
        wd_addr_dec(27) <= '1' ;
      when others   =>
        wd_addr_dec(27) <= '0' ;
    end case;
    case wd_addr is
      when "11100"  =>
        wd_addr_dec(28) <= '1' ;
      when others   =>
        wd_addr_dec(28) <= '0' ;
    end case;
    case wd_addr is
      when "11101"  =>
        wd_addr_dec(29) <= '1' ;
      when others   =>
        wd_addr_dec(29) <= '0' ;
    end case;
    case wd_addr is
      when "11110"  =>
        wd_addr_dec(30) <= '1' ;
      when others   =>
        wd_addr_dec(30) <= '0' ;
    end case;
    case wd_addr is
      when "11111"  =>
        wd_addr_dec(31) <= '1' ;
      when others   =>
        wd_addr_dec(31) <= '0' ;
    end case;
  end process wd_dec;


end em_intf;
```

```
--------------------------------------------------------------------------
--compass compile_off


configuration em_intf_CON of em_intf is
  for em_intf
  end for;
end em_intf_CON;
--compass compile_on

--------------------------------------------------------------------------
```

```
-- Author:  George Phan
-- Date of netlist generation:  Mar-15-96
-- controller for emulator process.
--------------------------------------------------------------------------

library IEEE;
-- library gsc1000d;
use IEEE.STD_LOGIC_1164.ALL;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.ALL;
--compass compile_off
use COMPASS_LIB.COMPASS_ETC.ALL;
--compass compile_on

entity em_ctlf is

--compass compile_off
--compass compile_on


    port ( ayk_ins : in Std_Logic_Vector(15 downto 0) := (others => 'U');
           rom_addr : out Std_Logic_Vector(8 downto 0) := (others => 'U');
           CLK : in Std_Logic := 'U';
           new_ppc_ins : out Std_Logic := 'U');
end em_ctlf;



--------------------------------------------------------------------------

architecture em_ctlf OF em_ctlf is
--compass dp_gates;

--    component ni01d2
--        port ( I : in Std_Logic;

 --              Z : out Std_Logic );
 --   end component;

   CONSTANT delay : time := 0.8 ns;

signal new_opaddr, new_opaddr_1, no_opaddr, next_iaddr, naddr :
               Std_Logic_Vector(8 downto 0) := (others => 'U');

signal new_opcyl, new_opcyl_1, no_opcyl, next_icyl, ncyl :
               Std_Logic_Vector(5 downto 0) := (others => 'U');

signal  sel_addr, sel_addri
              : Std_Logic := 'U';

Signal icycle, icle_dec
     :  Std_Logic_Vector(5 downto 0) := (others => 'U');



begin

   -- tielowninv: ni01d2
   --     port map ( I => vss, Z => tielow );


reg1:
```



```
process(clk, next_iaddr,
 new_opaddr, new_opaddr_1, no_opaddr, next_iaddr, naddr,
 new_opcyl, new_opcyl_1, no_opcyl, next_icyl, ncyl ,
 sel_addr, sel_addri,
 icycle, icle_dec)

begin
  If (rising_edge(clk)) THEN
       rom_addr <= next_iaddr(8 downto 0) after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
      rom_addr <= "XXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS reg1;

  rom_addr_sel:
  PROCESS(new_opaddr, new_opaddr_1, no_opaddr, clk, sel_addr)
  BEGIN
    if (clk ='1') then
      next_iaddr <= new_opaddr;
    Elsif (sel_addr ='0') then
      next_iaddr <= no_opaddr;
    ELSE
      next_iaddr <= new_opaddr_1;
    end if;
  END PROCESS rom_addr_sel;

  rom_cyl_sel:
  PROCESS(new_opcyl, new_opcyl_1, no_opcyl, clk, sel_addr)
  BEGIN
    if (clk ='1') then
      next_icyl <= new_opcyl;
    Elsif (sel_addr = '0') then
      next_icyl <= no_opcyl;
    ELSE
      next_icyl <= new_opcyl_1;
    end if;
  END PROCESS rom_cyl_sel;

  zerodet:
  PROCESS( next_icyl)
  BEGIN
      sel_addri <= next_icyl(5) or next_icyl(4) or next_icyl(3) or next_icyl(2) or next_icyl(1)
      new_ppc_ins <= next_icyl(5) or next_icyl(4) or next_icyl(3) or next_icyl(2) or next_icyl(
  END PROCESS zerodet;

  cycle_dec:
  PROCESS( ncyl)
  BEGIN
      new_opcyl_1 <=  ncyl - '1' ;
  END PROCESS cycle_dec;

  iaddr_inc:
  PROCESS( naddr)
  BEGIN
      new_opaddr_1 <=  naddr + '1';
  END PROCESS iaddr_inc;

reg2:
process(clk, sel_addri)
begin
```

```
  If (rising_edge(clk)) THEN
       sel_addr <= sel_addri after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
      sel_addr <= 'X';
--compass compile_on
    END IF;
  END PROCESS reg2;

reg3:
process(clk, next_iaddr)
begin

  If (rising_edge(clk)) THEN
         naddr <= next_iaddr after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
        naddr <= "XXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS reg3;

reg4:
process(clk, next_icyl)
begin

  If (rising_edge(clk)) THEN
         ncyl <= next_icyl after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
        ncyl <= "XXXXXX";
--compass compile_on
    END IF;
  END PROCESS reg4;



  op_cycle_decode:
  PROCESS(ayk_ins)
  BEGIN
    CASE ayk_ins(15 downto 8) IS
---- 10/10/27---
      WHEN "00010000" =>
        new_opcyl <= "001010";
        new_opaddr <= "000011011";
---- 14/5/38---
      WHEN "00010100" =>
        new_opcyl <= "000101";
        new_opaddr <= "000100110";
---- 18/4/44---
      WHEN "00011000" =>
        new_opcyl <= "000100";
        new_opaddr <= "000101100";
---- 20/6/49---
      WHEN "00100000" =>
        new_opcyl <= "000110";
        new_opaddr <= "000110001";
---- 22/6/49---
      WHEN "00100010" =>
        new_opcyl <= "000110";
        new_opaddr <= "000110001";
---- 28/6/49---
      WHEN "00101000" =>
        new_opcyl <= "000110";
```



```
        new_opaddr <= "000110001";
---- 2A/6/49---
      WHEN "00101010" =>
        new_opcyl <= "000110";
        new_opaddr <= "000110001";
---- 24/6/56---
      WHEN "00100100" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111000";
---- 26/6/56---
      WHEN "00100110" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111000";
---- 2C/6/56---
      WHEN "00101100" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111000";
---- 2E/6/56---
      WHEN "00101110" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111000";
---- 30/6/63---
      WHEN "00110000" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111111";
---- 32/6/63---
      WHEN "00110010" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111111";
---- 38/6/63---
      WHEN "00111000" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111111";
---- 3A/6/63---
      WHEN "00111010" =>
        new_opcyl <= "000110";
        new_opaddr <= "000111111";
      WHEN "00000111" =>
---- 34/8/70---
      WHEN "00110100" =>
        new_opcyl <= "001000";
        new_opaddr <= "001000110";
---- 36/8/70---
      WHEN "00110110" =>
        new_opcyl <= "001000";
        new_opaddr <= "001000110";
---- 3C/6/79---
      WHEN "00111100" =>
        new_opcyl <= "000110";
        new_opaddr <= "001001111";
---- 3E/6/79---
      WHEN "00111110" =>
        new_opcyl <= "000110";
        new_opaddr <= "001001111";
---- 40/7/86---
      WHEN "01000000" =>
        new_opcyl <= "000111";
        new_opaddr <= "001010110";
---- 41/7/86---
      WHEN "01000001" =>
        new_opcyl <= "000111";
        new_opaddr <= "001010110";
---- 42/7/86---
      WHEN "01000010" =>
```

```
        new_opcyl <= "000111";
        new_opaddr <= "001010110";
---- 43/7/86---
      WHEN "01000011" =>
        new_opcyl <= "000111";
        new_opaddr <= "001010110";
---- 44/7---
      WHEN "01000100" =>
        new_opcyl <= "000111";
        new_opaddr <= "001010110";
---- 45/7---
      WHEN "01000101" =>
        new_opcyl <= "000111";
        new_opaddr <= "001010110";
---- 47/7---
      WHEN "01000111" =>
        new_opcyl <= "000111";
        new_opaddr <= "001010110";
---- 48/7/94---
      WHEN "01001000" =>
        new_opcyl <= "000111";
        new_opaddr <= "001011110";
---- 49/7/94---
      WHEN "01001001" =>
        new_opcyl <= "000111";
        new_opaddr <= "001011110";
---- 4A/7/94---
      WHEN "01001010" =>
        new_opcyl <= "000111";
        new_opaddr <= "001011110";
---- 4B/7/94---
      WHEN "01001011" =>
        new_opcyl <= "000111";
        new_opaddr <= "001011110";
---- 4C/7/94---
      WHEN "01001100" =>
        new_opcyl <= "000111";
        new_opaddr <= "001011110";
---- 4D/7/94---
      WHEN "01001101" =>
        new_opcyl <= "000111";
        new_opaddr <= "001011110";
---- 4F/7/94---
      WHEN "01001111" =>
        new_opcyl <= "000111";
        new_opaddr <= "001011110";
---- 50/5/102---
      WHEN "01010000" =>
        new_opcyl <= "000101";
        new_opaddr <= "001100110";
---- 51/5/102---
      WHEN "01010001" =>
        new_opcyl <= "000101";
        new_opaddr <= "001100110";
---- 52/5/102---
      WHEN "01010010" =>
        new_opcyl <= "000101";
        new_opaddr <= "001100110";
---- 53/5/102---
      WHEN "01010011" =>
        new_opcyl <= "000101";
        new_opaddr <= "001100110";
---- 54/5/108---
      WHEN "01010100" =>
```



```
        new_opcyl <= "000101";
        new_opaddr <= "001101100";
---- 55/5/108---
      WHEN "01010101" =>
        new_opcyl <= "000101";
        new_opaddr <= "001101100";
---- 57/5/108---
      WHEN "01010111" =>
        new_opcyl <= "000101";
        new_opaddr <= "001101100";
---- 58/6/114---
      WHEN "01011000" =>
        new_opcyl <= "000110";
        new_opaddr <= "001110010";
---- 59/6-/114--
      WHEN "01011001" =>
        new_opcyl <= "000110";
        new_opaddr <= "001110010";
---- 5A/6/114---
      WHEN "01011010" =>
        new_opcyl <= "000110";
        new_opaddr <= "001110010";
---- 5B/6/114---
      WHEN "01011011" =>
        new_opcyl <= "000110";
        new_opaddr <= "001110010";
---- 5C/7/121---
      WHEN "01011100" =>
        new_opcyl <= "000111";
        new_opaddr <= "001111001";
---- 5D/7/121---
      WHEN "01011101" =>
        new_opcyl <= "000111";
        new_opaddr <= "001111001";
---- 5E/7/121---
      WHEN "01011110" =>
        new_opcyl <= "000111";
        new_opaddr <= "001111001";
---- 5F/7/121---
      WHEN "01011111" =>
        new_opcyl <= "000111";
        new_opaddr <= "001111001";
---- 60/6/129---
      WHEN "01100000" =>
        new_opcyl <= "000110";
        new_opaddr <= "010000001";
---- 61/6/129---
      WHEN "01100001" =>
        new_opcyl <= "000110";
        new_opaddr <= "010000001";
---- 62/6/129---
      WHEN "01100010" =>
        new_opcyl <= "000110";
        new_opaddr <= "010000001";
---- 63/6/129---
      WHEN "01100011" =>
        new_opcyl <= "000110";
        new_opaddr <= "010000001";
---- 64/6/136---
      WHEN "01100100" =>
        new_opcyl <= "000110";
        new_opaddr <= "010001000";
---- 65/6/136---
      WHEN "01100101" =>
```

```
        new_opcyl <= "000110";
        new_opaddr <= "010001000";
---- 66/6/136---
      WHEN "01100110" =>
        new_opcyl <= "000110";
        new_opaddr <= "010001000";
---- 67/6/136---
      WHEN "01100111" =>
        new_opcyl <= "000110";
        new_opaddr <= "010001000";
---- 68/6/143---
      WHEN "01101000" =>
        new_opcyl <= "000110";
        new_opaddr <= "010001111";
---- 69/6/143---
      WHEN "01101001" =>
        new_opcyl <= "000110";
        new_opaddr <= "010001111";
---- 6A/6/143---
      WHEN "01101010" =>
        new_opcyl <= "000110";
        new_opaddr <= "010001111";
---- 6B/6/143---
      WHEN "01101011" =>
        new_opcyl <= "000110";
        new_opaddr <= "010001111";
---- 6C/6/149---
      WHEN "01101100" =>
        new_opcyl <= "001011";
        new_opaddr <= "010010101";
---- 6D/6/149---
      WHEN "01101101" =>
        new_opcyl <= "001011";
        new_opaddr <= "010010101";
---- 6E/6/149---
      WHEN "01101110" =>
        new_opcyl <= "001011";
        new_opaddr <= "010010101";
---- 6F/6/149---
      WHEN "01101111" =>
        new_opcyl <= "001011";
        new_opaddr <= "010010101";
---- 70/8/161---
      WHEN "01110000" =>
        new_opcyl <= "001000";
        new_opaddr <= "010100001";
---- 71/8/161---
      WHEN "01110001" =>
        new_opcyl <= "001000";
        new_opaddr <= "010100001";
---- 72/8/161---
      WHEN "01110010" =>
        new_opcyl <= "001000";
        new_opaddr <= "010100001";
---- 73/8/161---
      WHEN "01110011" =>
        new_opcyl <= "001000";
        new_opaddr <= "010100001";
---- A0/6/170---
      WHEN "10100000" =>
        new_opcyl <= "000110";
        new_opaddr <= "010101010";
---- A1/6/170---
      WHEN "10100001" =>
```



```
        new_opcyl <= "000110";
        new_opaddr <= "010101010";
---- A3/6/170---
      WHEN "10100011" =>
        new_opcyl <= "000110";
        new_opaddr <= "010101010";
---- A4/6/177---
      WHEN "10100100" =>
        new_opcyl <= "000110";
        new_opaddr <= "010110001";
---- A5/6/177---
      WHEN "10100101" =>
        new_opcyl <= "000110";
        new_opaddr <= "010110001";
---- A7/6/177---
      WHEN "10100111" =>
        new_opcyl <= "000110";
        new_opaddr <= "010110001";
---- A8/6/184---
      WHEN "10101000" =>
        new_opcyl <= "000110";
        new_opaddr <= "010111000";
---- A9/6/184---
      WHEN "10101001" =>
        new_opcyl <= "000110";
        new_opaddr <= "010111000";
---- AB/6/184---
      WHEN "10101011" =>
        new_opcyl <= "000110";
        new_opaddr <= "010111000";
---- AC/6/191---
      WHEN "10101100" =>
        new_opcyl <= "000110";
        new_opaddr <= "010111111";
---- AD/6/191---
      WHEN "10101101" =>
        new_opcyl <= "000110";
        new_opaddr <= "010111111";
---- AF/6/191---
      WHEN "10101111" =>
        new_opcyl <= "000110";
        new_opaddr <= "010111111";
---- B8/8/198---
      WHEN "10111000" =>
        new_opcyl <= "001000";
        new_opaddr <= "011000110";
---- B9/8/198--
      WHEN "10111001" =>
        new_opcyl <= "001000";
        new_opaddr <= "011000110";
---- BB/8/198---
      WHEN "10111011" =>
        new_opcyl <= "001000";
        new_opaddr <= "011000110";
---- BC/15/207---
      WHEN "10111100" =>
        new_opcyl <= "001111";
        new_opaddr <= "011001111";
---- BD/15/207---
      WHEN "10111101" =>
        new_opcyl <= "001111";
        new_opaddr <= "011001111";
---- BF/15/207---
      WHEN "10111111" =>
```

```
        new_opcyl <= "001111";
        new_opaddr <= "011001111";
---- C0/7/223---
      WHEN "11000000" =>
        new_opcyl <= "000111";
        new_opaddr <= "011011111";
---- C1/7/231---
      WHEN "11000001" =>
        new_opcyl <= "000111";
        new_opaddr <= "011100111";
---- C2/7/223---
      WHEN "11000010" =>
        new_opcyl <= "000111";
        new_opaddr <= "011011111";
---- C3/7/231---
      WHEN "11000011" =>
        new_opcyl <= "000111";
        new_opaddr <= "011100111";
---- C4/7/239---
      WHEN "11000100" =>
        new_opcyl <= "000111";
        new_opaddr <= "011101111";
---- C5/9/247---
      WHEN "11000101" =>
        new_opcyl <= "001001";
        new_opaddr <= "011110111";
---- C6/7/239---
      WHEN "11000110" =>
        new_opcyl <= "000111";
        new_opaddr <= "011101111";
---- C7/7/257---
      WHEN "11000111" =>
        new_opcyl <= "000111";
        new_opaddr <= "100000001";
---- C8/6/265---
      WHEN "11001000" =>
        new_opcyl <= "000110";
        new_opaddr <= "100001001";
---- C9/6/272---
      WHEN "11001001" =>
        new_opcyl <= "000110";
        new_opaddr <= "100010000";
---- CA/8/279---
      WHEN "11001010" =>
        new_opcyl <= "001000";
        new_opaddr <= "100010111";
---- CB/8/279---
      WHEN "11001011" =>
        new_opcyl <= "001000";
        new_opaddr <= "100010111";
---- CD/4/288---
      WHEN "11001101" =>
        new_opcyl <= "000100";
        new_opaddr <= "100100000";
---- CE/8/293---
      WHEN "11001110" =>
        new_opcyl <= "001000";
        new_opaddr <= "100100101";
---- CF/8/301---
      WHEN "11001111" =>
        new_opcyl <= "001000";
        new_opaddr <= "100101101";
      WHEN OTHERS =>
        new_opcyl <= "000000";
```



```
        new_opaddr <= "000000000";
    END CASE;
  END PROCESS op_cycle_decode;


end em_ctlf;

--------------------------------------------------------------------------------
--compass compile_off


configuration em_ctlf_CON of em_ctlf is
  for em_ctlf
  end for;
end em_ctlf_CON;
--compass compile_on

--------------------------------------------------------------------------------
```

```
-- ***************************************************
-- This vhdl code is for AN/AYK instruction format decoder
-- It will read 8 bit input from the instruction and decode
-- the format and derived 16 bit ra and rm
-- ***************************************************


library IEEE;
-- library gsc1000d;
use IEEE.STD_LOGIC_1164.ALL;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.ALL;
--compass compile_off
use COMPASS_LIB.COMPASS_ETC.ALL;
--compass compile_on

ENTITY fmatdeco IS
--compass compile_off
--compass compile_on
   PORT (inputdata   : IN std_logic_vector(15 downto 0);
         SR114       : IN std_logic                    ;
         SR2HB       : IN std_logic_vector(7  downto 0);
         ydata       : IN std_logic_vector(15 downto 0);
         npcount     : IN std_logic_vector(15 downto 0);
         IW1data     : IN std_logic_vector(15 downto 0);
         IW2data     : IN std_logic_vector(15 downto 0);
         IW2ddata    : IN std_logic_vector(15 downto 0);
         ayk_dcyl    : IN std_logic ;
         ayk_ycyl    : IN std_logic ;
         ayk_w1cyl   : IN std_logic ;
         ayk_w2cyl   : IN std_logic ;
         ayk_w2dcyl  : IN std_logic ;
         ayk_ecyl    : IN std_logic ;
         clk         : IN std_logic ;
         sins        : IN std_logic ;
         dins        : IN std_logic ;
         bins        : IN std_logic ;
         rm_data     : IN   std_logic_vector(15 downto 0);
         odd         : OUT std_logic                    ;
         ra_addr     : OUT std_logic_vector(31 downto 0);
         rm_addr     : OUT std_logic_vector(31 downto 0);
          a_addr     : OUT std_logic_vector(4  downto 0);
          m_addr     : OUT std_logic_vector(4  downto 0);
        mem_addr     : OUT std_logic_vector(15 downto 0));
END fmatdeco ;


ARCHITECTURE fmatdeco  OF fmatdeco IS
--compass dp_gates;

signal y_addr : std_logic_vector (15 downto 0);
signal iw1_addr : std_logic_vector (15 downto 0);
signal iw2_addr : std_logic_vector (15 downto 0);
signal iw2d_addr : std_logic_vector (15 downto 0);
signal m1 : std_logic_vector (3 downto 0);
signal maddr : std_logic_vector (3 downto 0);
signal rmaddr : std_logic_vector (4 downto 0);
signal rm_daddr : std_logic_vector (31 downto 0);
signal aaddr : std_logic_vector (3 downto 0);
signal raaddr : std_logic_vector (4 downto 0);
signal ra_daddr : std_logic_vector (31 downto 0);
signal data_addr : std_logic_vector (15 downto 0);
signal rm_register : std_logic_vector (15 downto 0);
```



```
signal rm1_register : std_logic_vector (15 downto 0);
signal rxx_register : std_logic_vector (15 downto 0);
signal rm_register0 : std_logic_vector (15 downto 0);
signal rm1_register0 : std_logic_vector (15 downto 0);
signal rxx_register0 : std_logic_vector (15 downto 0);
signal pcount : std_logic_vector (15 downto 0);
signal pcount0 : std_logic_vector (15 downto 0);
signal pcount1 : std_logic_vector (15 downto 0);
signal pcount2 : std_logic_vector (15 downto 0);
signal iw1_tmp8 : std_logic_vector (15 downto 0);
signal iw1_tmpa : std_logic_vector (15 downto 0);
signal iw1_tmpc : std_logic_vector (15 downto 0);
signal iw1_tmpe : std_logic_vector (15 downto 0);
signal iw1y : std_logic_vector (15 downto 0);
signal iw2y : std_logic_vector (15 downto 0);
constant delay : time := 0.8 ns;


  alias J : std_logic_vector(3 downto 0) is IW1data(15 downto 12);
  alias x : std_logic_vector(3 downto 0) is IW1data(3 downto 0);

  alias Rm : std_logic_vector(15 downto 0) is Rm_Register(15 downto 0);
  alias Rxx : std_logic_vector(15 downto 0) is Rxx_Register(15 downto 0);
  alias Rm1 : std_logic_vector(15 downto 0) is Rm1_Register(15 downto 0);

  alias aa : std_logic_vector(3 downto 0) is inputdata(7 downto 4);
  alias m : std_logic_vector(3 downto 0) is inputdata(3 downto 0);

BEGIN

reg1:
process(clk, rm_register0,
  y_addr ,
  iw1_addr ,
  iw2_addr ,
  iw2d_addr ,
  m1 ,
  maddr ,
  rmaddr ,
  rm_daddr ,
  aaddr ,
  raaddr ,
  ra_daddr ,
  data_addr ,
  rm_register ,
  rm1_register ,
  rxx_register ,
  rm1_register0 ,
  rxx_register0 ,
  pcount ,
  pcount0 ,
  pcount1 ,
  pcount2 ,
  iw1_tmp8 ,
  iw1_tmpa ,
  iw1_tmpc ,
  iw1_tmpe ,
  iw1y ,
  iw2y )
begin

  If (rising_edge(clk)) THEN
       rm_register <= rm_register0  after delay;
--compass compile_off
```

```
    ELSIF (To_X01(clk) = 'X') THEN
      rm_register <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS reg1;

rmux1:
process(ayk_dcy1, rm_register, rm_data)
begin
case ayk_dcy1 is
  when '0'  =>
   rm_register0 <= rm_register ;
  when others  =>
   rm_register0 <= rm_data ;
 END CASE;
end process rmux1;
reg3:
process(clk, rxx_register0)
begin

  If (rising_edge(clk)) THEN
       rxx_register <= rxx_register0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
      rxx_register <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS reg3;

rmux3:
process(ayk_w2cyl, rxx_register, rm_data)
begin
case ayk_w2cyl is
  when '0'  =>
   rxx_register0 <= rxx_register ;
  when others  =>
   rxx_register0 <= rm_data ;
 END CASE;
end process rmux3;
reg2:
process(clk, rm1_register0)
begin

  If (rising_edge(clk)) THEN
       rm1_register <= rm1_register0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
      rm1_register <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS reg2;

rmux2:
process(ayk_ycyl, rm1_register, rm_data)
begin
case ayk_ycyl is
  when '0'  =>
   rm1_register0 <= rm1_register ;
  when others  =>
   rm1_register0 <= rm_data ;
 END CASE;
end process rmux2;

Radd_decoder:
```



```
PROCESS (inputdata,ayk_dcy1,rm_data,rm_register, rm1_register, rxx_register,
ayk_ycy1,ayk_w1cy1, ayk_w2cy1,ayk_w2dcy1,ayk_ecy1,aaddr,iw1data,
rmaddr,sr114,maddr,raaddr)

    BEGIN
    odd <= inputdata(0) ;
       m1 <= m + '1' ;
    case ayk_dcy1  is
    when '1'  =>
    maddr <= m  ;
    when others =>
    maddr <= "0000"  ;
    end case;
    case ayk_ycy1  is
    when '1'  =>
    maddr <= m1  ;
    when others =>
    maddr <= "0000"  ;
    end case;
    case ayk_w2cyl  is
    when '1'  =>
    maddr <= x   ;
    when others =>
    maddr <= "0000"  ;
    end case;

   m_addr <= sr114 & maddr;
   rmaddr <= sr114 & maddr;
    case rmaddr is
       when "00000"  =>
         rm_daddr(0) <= '1' ;
       when others   =>
         rm_daddr(0) <= '0' ;
     end case;
    case rmaddr is
       when "00001"  =>
         rm_daddr(1) <= '1' ;
       when others   =>
         rm_daddr(1) <= '0' ;
     end case;
    case rmaddr is
       when "00010"  =>
         rm_daddr(2) <= '1' ;
       when others   =>
         rm_daddr(2) <= '0' ;
     end case;
    case rmaddr is
       when "00011"  =>
         rm_daddr(3) <= '1' ;
       when others   =>
         rm_daddr(3) <= '0' ;
     end case;
    case rmaddr is
       when "00100"  =>
         rm_daddr(4) <= '1' ;
       when others   =>
         rm_daddr(4) <= '0' ;
     end case;
    case rmaddr is
       when "00101"  =>
         rm_daddr(5) <= '1' ;
       when others   =>
         rm_daddr(5) <= '0' ;
     end case;
```

```
case rmaddr is
   when "00110"  =>
     rm_daddr(6) <= '1' ;
   when others   =>
     rm_daddr(6) <= '0' ;
 end case;
case rmaddr is
   when "00111"  =>
     rm_daddr(7) <= '1' ;
   when others   =>
     rm_daddr(7) <= '0' ;
 end case;
case rmaddr is
   when "01000"  =>
     rm_daddr(8) <= '1' ;
   when others   =>
     rm_daddr(8) <= '0' ;
 end case;
case rmaddr is
   when "01001"  =>
     rm_daddr(9) <= '1' ;
   when others   =>
     rm_daddr(9) <= '0' ;
 end case;
case rmaddr is
   when "01010"  =>
     rm_daddr(10) <= '1' ;
   when others   =>
     rm_daddr(10) <= '0' ;
 end case;
case rmaddr is
   when "01011"  =>
     rm_daddr(11) <= '1' ;
   when others   =>
     rm_daddr(11) <= '0' ;
 end case;
case rmaddr is
   when "01100"  =>
     rm_daddr(12) <= '1' ;
   when others   =>
     rm_daddr(12) <= '0' ;
 end case;
case rmaddr is
   when "01101"  =>
     rm_daddr(13) <= '1' ;
   when others   =>
     rm_daddr(13) <= '0' ;
 end case;
case rmaddr is
   when "01110"  =>
     rm_daddr(14) <= '1' ;
   when others   =>
     rm_daddr(14) <= '0' ;
 end case;
case rmaddr is
   when "01111"  =>
     rm_daddr(15) <= '1' ;
   when others   =>
     rm_daddr(15) <= '0' ;
 end case;
case rmaddr is
   when "10000"  =>
     rm_daddr(16) <= '1' ;
   when others   =>
```



```
     rm_daddr(16) <= '0' ;
 end case;
case rmaddr is
   when "10001"  =>
     rm_daddr(17) <= '1' ;
   when others   =>
     rm_daddr(17) <= '0' ;
 end case;
case rmaddr is
   when "10010"  =>
     rm_daddr(18) <= '1' ;
   when others   =>
     rm_daddr(18) <= '0' ;
 end case;
case rmaddr is
   when "10011"  =>
     rm_daddr(19) <= '1' ;
   when others   =>
     rm_daddr(19) <= '0' ;
 end case;
case rmaddr is
   when "10100"  =>
     rm_daddr(20) <= '1' ;
   when others   =>
     rm_daddr(20) <= '0' ;
 end case;
case rmaddr is
   when "10101"  =>
     rm_daddr(21) <= '1' ;
   when others   =>
     rm_daddr(21) <= '0' ;
 end case;
case rmaddr is
   when "10110"  =>
     rm_daddr(22) <= '1' ;
   when others   =>
     rm_daddr(22) <= '0' ;
 end case;
case rmaddr is
   when "10111"  =>
     rm_daddr(23) <= '1' ;
   when others   =>
     rm_daddr(23) <= '0' ;
 end case;
case rmaddr is
   when "11000"  =>
     rm_daddr(24) <= '1' ;
   when others   =>
     rm_daddr(24) <= '0' ;
 end case;
case rmaddr is
   when "11001"  =>
     rm_daddr(25) <= '1' ;
   when others   =>
     rm_daddr(25) <= '0' ;
 end case;
case rmaddr is
   when "11010"  =>
     rm_daddr(26) <= '1' ;
   when others   =>
     rm_daddr(26) <= '0' ;
 end case;
case rmaddr is
   when "11011"  =>
```

```
        rm_daddr(27) <= '1' ;
      when others   =>
        rm_daddr(27) <= '0' ;
   end case;
  case rmaddr is
     when "11100"  =>
        rm_daddr(28) <= '1' ;
     when others   =>
        rm_daddr(28) <= '0' ;
   end case;
  case rmaddr is
     when "11101"  =>
        rm_daddr(29) <= '1' ;
     when others   =>
        rm_daddr(29) <= '0' ;
   end case;
  case rmaddr is
     when "11110"  =>
        rm_daddr(30) <= '1' ;
     when others   =>
        rm_daddr(30) <= '0' ;
   end case;
  case rmaddr is
     when "11111"  =>
        rm_daddr(31) <= '1' ;
     when others   =>
        rm_daddr(31) <= '0' ;
   end case;

 case ayk_dcyl is
  when '1'  =>
  aaddr <= aa ;
  when others  =>
  aaddr <= "XXXX" ;
   end case;

 a_addr <= sr114 & aaddr;
 raaddr <= sr114 & aaddr;
  case raaddr is
     when "00000"  =>
        ra_daddr(0) <= '1' ;
     when others   =>
        ra_daddr(0) <= '0' ;
   end case;
  case raaddr is
     when "00001"  =>
        ra_daddr(1) <= '1' ;
     when others   =>
        ra_daddr(1) <= '0' ;
   end case;
  case raaddr is
     when "00010"  =>
        ra_daddr(2) <= '1' ;
     when others   =>
        ra_daddr(2) <= '0' ;
   end case;
  case raaddr is
     when "00011"  =>
        ra_daddr(3) <= '1' ;
     when others   =>
        ra_daddr(3) <= '0' ;
   end case;
  case raaddr is
     when "00100"  =>
```



```
        ra_daddr(4) <= '1' ;
      when others   =>
        ra_daddr(4) <= '0' ;
   end case;
  case raaddr is
     when "00101"  =>
        ra_daddr(5) <= '1' ;
     when others   =>
        ra_daddr(5) <= '0' ;
   end case;
  case raaddr is
     when "00110"  =>
        ra_daddr(6) <= '1' ;
     when others   =>
        ra_daddr(6) <= '0' ;
   end case;
  case raaddr is
     when "00111"  =>
        ra_daddr(7) <= '1' ;
     when others   =>
        ra_daddr(7) <= '0' ;
   end case;
  case raaddr is
     when "01000"  =>
        ra_daddr(8) <= '1' ;
     when others   =>
        ra_daddr(8) <= '0' ;
   end case;
  case raaddr is
     when "01001"  =>
        ra_daddr(9) <= '1' ;
     when others   =>
        ra_daddr(9) <= '0' ;
   end case;
  case raaddr is
     when "01010"  =>
        ra_daddr(10) <= '1' ;
     when others   =>
        ra_daddr(10) <= '0' ;
   end case;
  case raaddr is
     when "01011"  =>
        ra_daddr(11) <= '1' ;
     when others   =>
        ra_daddr(11) <= '0' ;
   end case;
  case raaddr is
     when "01100"  =>
        ra_daddr(12) <= '1' ;
     when others   =>
        ra_daddr(12) <= '0' ;
   end case;
  case raaddr is
     when "01101"  =>
        ra_daddr(13) <= '1' ;
     when others   =>
        ra_daddr(13) <= '0' ;
   end case;
  case raaddr is
     when "01110"  =>
        ra_daddr(14) <= '1' ;
     when others   =>
        ra_daddr(14) <= '0' ;
   end case;
```

```
case raaddr is
   when "01111"  =>
     ra_daddr(15) <= '1' ;
   when others   =>
     ra_daddr(15) <= '0' ;
 end case;
case raaddr is
   when "10000"  =>
     ra_daddr(16) <= '1' ;
   when others   =>
     ra_daddr(16) <= '0' ;
 end case;
case raaddr is
   when "10001"  =>
     ra_daddr(17) <= '1' ;
   when others   =>
     ra_daddr(17) <= '0' ;
 end case;
case raaddr is
   when "10010"  =>
     ra_daddr(18) <= '1' ;
   when others   =>
     ra_daddr(18) <= '0' ;
 end case;
case raaddr is
   when "10011"  =>
     ra_daddr(19) <= '1' ;
   when others   =>
     ra_daddr(19) <= '0' ;
 end case;
case raaddr is
   when "10100"  =>
     ra_daddr(20) <= '1' ;
   when others   =>
     ra_daddr(20) <= '0' ;
 end case;
case raaddr is
   when "10101"  =>
     ra_daddr(21) <= '1' ;
   when others   =>
     ra_daddr(21) <= '0' ;
 end case;
case raaddr is
   when "10110"  =>
     ra_daddr(22) <= '1' ;
   when others   =>
     ra_daddr(22) <= '0' ;
 end case;
case raaddr is
   when "10111"  =>
     ra_daddr(23) <= '1' ;
   when others   =>
     ra_daddr(23) <= '0' ;
 end case;
case raaddr is
   when "11000"  =>
     ra_daddr(24) <= '1' ;
   when others   =>
     ra_daddr(24) <= '0' ;
 end case;
case raaddr is
   when "11001"  =>
     ra_daddr(25) <= '1' ;
   when others   =>
```



```
     ra_daddr(25) <= '0' ;
 end case;
case raaddr is
   when "11010"  =>
     ra_daddr(26) <= '1' ;
   when others   =>
     ra_daddr(26) <= '0' ;
 end case;
case raaddr is
   when "11011"  =>
     ra_daddr(27) <= '1' ;
   when others   =>
     ra_daddr(27) <= '0' ;
 end case;
case raaddr is
   when "11100"  =>
     ra_daddr(28) <= '1' ;
   when others   =>
     ra_daddr(28) <= '0' ;
 end case;
case raaddr is
   when "11101"  =>
     ra_daddr(29) <= '1' ;
   when others   =>
     ra_daddr(29) <= '0' ;
 end case;
case raaddr is
   when "11110"  =>
     ra_daddr(30) <= '1' ;
   when others   =>
     ra_daddr(30) <= '0' ;
 end case;
case raaddr is
   when "11111"  =>
     ra_daddr(31) <= '1' ;
   when others   =>
     ra_daddr(31) <= '0' ;
 end case;
end process radd_decoder;
-- Begin decode for address format

preg:
process(clk, pcount0)
begin

  If (rising_edge(clk)) THEN
      pcount <= pcount0  after delay;
--compass compile_off
    ELSIF (To_X01(clk) = 'X') THEN
      pcount <= "XXXXXXXXXXXXXXXX";
--compass compile_on
    END IF;
  END PROCESS preg;

npc:
process (pcount)
begin
      pcount1 <= pcount + '1';
      pcount2 <= pcount + "10";
end process npc;

mem_addr_decoder:
process (inputdata, iw1data,iw2data,ydata,sr2hb,ayk_dcyl,ayk_ycyl,ayk_w1cyl,
        ayk_w2cyl, ayk_w2dcyl, ayk_ecyl,pcount1,pcount2,npcount,sins,dins, bins,
```

```
data_addr,iw2d_addr,iwly,iw2ddata,rxx_register,rm_register,rm1_register,
iw1_addr,iw1_tmp8,iw1_tmpa,iw1_tmpc,iw1_tmpe,iw2y,pcount,rm_data)
begin
     case sins   is
        when '1'  =>
          pcount0 <= pcount1 ;
        when others   =>
          pcount0 <= "XXXXXXXXXXXXXXXX" ;
      end case;
     case dins   is
        when '1'  =>
          pcount0 <= pcount2 ;
        when others   =>
          pcount0 <= "XXXXXXXXXXXXXXXX" ;
      end case;
     case bins   is
        when '1'  =>
          pcount0 <= npcount ;
        when others   =>
          pcount0 <= "XXXXXXXXXXXXXXXX" ;
      end case;

      case sr2hb(1 downto 0) is
           when "10"          =>
           iw1_tmp8 <= ydata;
           iwly     <= "0000000000000000";
           when "11"          =>
           iw1_tmp8 <= ydata + rm;
           iwly     <= "0000000000000000";
           when others        =>
           iw1_tmp8 <= "0000000000000000";
           iwly     <= rm;
       end case;
      case sr2hb(3 downto 2) is
           when "10"          =>
           iw1_tmpa <= ydata;
           iwly     <= "0000000000000000";
           when "11"          =>
           iw1_tmpa <= ydata + rm;
           iwly     <= "0000000000000000";
           when others        =>
           iw1_tmpa <= "0000000000000000";
           iwly     <= rm;
       end case;
      case sr2hb(5 downto 4) is
           when "10"          =>
           iw1_tmpc <= ydata;
           iwly     <= "0000000000000000";
           when "11"          =>
           iw1_tmpc <= ydata + rm;
           iwly     <= "0000000000000000";
           when others        =>
           iw1_tmpc <= "0000000000000000";
           iwly     <= rm;
       end case;
      case sr2hb(7 downto 6) is
           when "10"          =>
           iw1_tmpe <= ydata;
           iwly     <= "0000000000000000";
           when "11"          =>
           iw1_tmpe <= ydata + rm;
           iwly     <= "0000000000000000";
           when others        =>
           iw1_tmpe <= "0000000000000000";
```



```
           iwly     <= rm;
      end case;
      y_addr <= pcount + '1';
       case m is
       when "1000"  =>
          iw1_addr  <= iw1_tmp8;
       when "1010"  =>
          iw1_addr  <= iw1_tmpa;
       when "1100"  =>
          iw1_addr  <= iw1_tmpc;
       when "1110"  =>
          iw1_addr  <= iw1_tmpe;
       when others  =>
          iw1_addr  <= "0000000000000000" ;
       end case;
       iw2_addr <= iw1_addr + '1';

            case J is
               when "0000" => iw2y <= iw2data ;
               when "0001" => iw2y <= iw2data + Rxx ;
               when "0010" => iw2y <= iw2data + Rm ;
               when "0011" => iw2y <= iw2data + Rm1 ;
               when "0100" => iw2d_addr <= iw2data ;  -- indirect word addressing
               when "0101" => iw2d_addr <= iw2data + Rxx ; -- indirect word addressing
               when "0110" => iw2d_addr <= iw2data + Rm ;  -- indirect word addressing
               when "0111" => iw2d_addr <= iw2data + Rm1 ; -- indirect word addressing
               when others => iw2y <= "0000000000000000" ;
                              iw2d_addr <= "0000000000000000" ;
            end case ;

      case m is
       when "0000"  =>
          y_addr   <= ydata;
       when "1000"  =>
          y_addr   <= iwly or iw2y or iw2ddata;
       when others  =>
          y_addr   <= ydata + Rm_data;
       end case;

      case ayk_ycyl is
       when '1'  =>
       data_addr   <= y_addr;
       when others  =>
       data_addr   <= "XXXXXXXXXXXXXXXX";
      end case;
      case ayk_w1cyl is
       when '1'  =>
       data_addr   <= iw1_addr;
       when others  =>
       data_addr   <= "XXXXXXXXXXXXXXXX";
      end case;
      case ayk_w2cyl is
       when '1'  =>
       data_addr   <= iw2_addr;
       when others  =>
       data_addr   <= "XXXXXXXXXXXXXXXX";
      end case;
      case ayk_w2dcyl is
       when '1'  =>
       data_addr   <= iw2d_addr;
       when others  =>
       data_addr   <= "XXXXXXXXXXXXXXXX";
      end case;
```

```
        mem_addr <= data_addr or pcount;


END PROCESS mem_addr_decoder;
end fmatdeco;

--compass compile_off

configuration fmatdeco_CON of fmatdeco is
  for fmatdeco
  end for;
end fmatdeco_CON;
--compass compile_on
-----------------------------------------------
```

```
-- ****************************************************
-- This vhdl code readin 16 bit instruction input
-- do decoding and output the control for program counter.
-- ****************************************************

--USE std.std_logic.ALL;
--USE std.std.ttl.ALL;
library IEEE;
use IEEE.std_logic_1164.all;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.all;
--compass compile_off
use COMPASS_LIB.COMPASS_ETC.ALL;
--compass compile_on

use Work.Emulator_datatype.all ;

ENTITY pccon  IS
   PORT (Inputdata : IN std_logic_vector(15 downto 10);
         m         : IN std_logic_vector(3 downto 0);
         opdeco_en : IN std_logic ;
         sins      : OUT std_logic;
         dins      : OUT std_logic;
         bins      : OUT std_logic;
         fpins     : OUT std_logic;
         orop      : OUT std_logic;
         andop     : OUT std_logic;
         xorop     : OUT std_logic;
         sr1op     : OUT std_logic;
         sr2op     : OUT std_logic;
         selmr     : OUT std_logic);

         -- LOAD/STORE
    signal      byte_load : std_logic ;
    signal      byte_load_index : std_logic ;
    signal      byte_store_index : std_logic ;
    signal      double_load_index : std_logic ;
    signal      literal_load : std_logic ;
    signal      load_multiple : std_logic ;
    signal      load_mult_reg : std_logic ;
    signal      store_mult_reg : std_logic ;
    signal      word_load_index : std_logic ;
    signal      move_block : std_logic ;
    signal      load : std_logic ;
    signal      load_bit : std_logic ;
    signal      load_double : std_logic ;
    signal      load_mult : std_logic ;
    signal      byte_store : std_logic ;
    signal      store : std_logic ;
    signal      store_double : std_logic ;
    signal      store_address : std_logic ;
    signal      store_zero : std_logic ;
    signal      decindex_store : std_logic ;
    signal      decindex_store_double : std_logic ;
    signal      load_store : std_logic ;

 -- SUPPORT CHANNEL
     signal    jump_busy : std_logic ;
     signal    jump_input : std_logic ;
signal  jump_io : std_logic ;
  -- ARITHMETIC
     signal   add : std_logic ;
     signal   add_byte : std_logic ;
```



```
     signal   add_double : std_logic ;
     signal   subtract : std_logic ;
     signal   subtract_byte : std_logic ;
     signal   subtract_double : std_logic ;
     signal   multiply : std_logic ;
     signal   multiply_double : std_logic ;
     signal   divide : std_logic ;
     signal   divide_double : std_logic ;
     signal   fpoint_add : std_logic ;
     signal   fpoint_divide : std_logic ;
     signal   fpoint_mult : std_logic ;
     signal   fpoint_subtract : std_logic ;
     signal   literal_add : std_logic ;
     signal   literal_double : std_logic ;
     signal   sign_ext_double : std_logic ;
     signal   sign_ext : std_logic ;

  -- LOGICAL
     signal   and_op : std_logic ;
     signal   or_op : std_logic ;
     signal   xor_op : std_logic ;

  -- COMPARE
     signal   byte_comp : std_logic ;
     signal   comp : std_logic ;
     signal   bit_comp : std_logic ;
     signal   comp_double : std_logic ;
     signal   logical_comp : std_logic ;
     signal   masked_comp : std_logic ;
     signal   literal_comp : std_logic ;
     signal   float_comp : std_logic ;


  -- SHIFT
     signal    alg_ldouble_shft : std_logic ;
     signal   alg_left_shft : std_logic ;
     signal   alg_right_shft : std_logic ;
     signal   alg_rdouble_shft : std_logic ;
     signal   cir_left_shft : std_logic ;
     signal   cir_ldouble_shft : std_logic ;
     signal   literal_left_shft : std_logic ;
     signal   literal_ldouble_shft : std_logic ;
     signal   literal_right_shft : std_logic ;
     signal   literal_rdouble_shft : std_logic ;
     signal   logical_right_shft : std_logic ;
     signal   logical_rdouble_shft : std_logic ;

  -- STACK/QUEUE
     signal     stack_get_top : std_logic ;
     signal     stack_put_top : std_logic ;
     signal     queue_get_top : std_logic ;
     signal     queue_put_top : std_logic ;
     signal     queue_put_bottom : std_logic ;

  -- Miscellaneous
     signal      float_to_single : std_logic ;
     signal      float_to_double : std_logic ;
     signal      fixed_to_float : std_logic ;
     signal      fixed_double_to_float : std_logic ;
     signal      floating_normalize : std_logic ;
     signal      algebraic_left_quadruple_shift : std_logic ;
     signal      algebraic_right_quadruple_shift : std_logic ;
     signal      executive_return : std_logic ;
     signal      count_ones : std_logic ;
```

```
    signal      biased_fetch : std_logic ;
    signal      masked_substitute : std_logic ;
    signal      remote_execute : std_logic ;
    signal      set_bit : std_logic ;
    signal      zero_bit : std_logic ;
    signal      reverse_register : std_logic ;
    signal      scale_factor : std_logic ;

-- CLOCK(executive mode)
    signal          disable_clock_int : std_logic ;
    signal          disable_rtc_register : std_logic ;
    signal          disable_monitor_clock : std_logic ;
    signal          enable_clock_int : std_logic ;
    signal          enable_clock_and_int : std_logic ;
    signal          load_rtc_lower : std_logic ;
    signal          load_double_and_clock : std_logic ;
    signal          load_enable_monitor_clock : std_logic ;
    signal          load_clock : std_logic ;
    signal          reset_bit_timer : std_logic ;
    signal          diagnostic_jump : std_logic ;

-- EXECUTIVE MODE
    signal         jump_stop_key : std_logic ;
    signal         jump_stop_key2 : std_logic ;
    signal         jump_after_stop : std_logic ;
    signal         load_addr_register : std_logic ;
    signal         load_physical_addr : std_logic ;
    signal         load_physical_location : std_logic ;
    signal         store_physical_location : std_logic ;
    signal         load_sr1 : std_logic ;
    signal         load_sr2 : std_logic ;
-- MISCELLANEOUS (EXECUTIVE MODE)
    signal          set_bit_indicator : std_logic ;
    signal          init_processor_int : std_logic ;
    signal          ipl_failed : std_logic ;

-- ***** SPECIAL HANDLE *****
    -- load/store
    signal          load_p : std_logic ;
    signal          store_clock : std_logic ;
    signal          store_clock_double : std_logic ;
    signal          store_sr1 : std_logic ;
    signal          store_sr2 : std_logic ;
    signal          store_monitor_clock : std_logic ;
    -- arthimatic
    signal          fixed_point_double_arccosine : std_logic ;
    signal          fixed_point_double_arcsine : std_logic ;
    signal          fixed_point_double_arctan : std_logic ;
    signal          fixed_point_double_exponent : std_logic ;
    signal          fixed_point_cross_product : std_logic ;
    signal          fixed_point_matrix_multiply : std_logic ;
    signal          fixed_point_double_natural_log : std_logic ;
    signal          fixed_point_sum_square : std_logic ;
    signal          fixed_point_double_sine_cosine : std_logic ;
    signal          fixed_point_double_square_root : std_logic ;
    signal          decrease_ra_1 : std_logic ;
    signal          decrease_ra_2 : std_logic ;
    signal          increase_ra_1 : std_logic ;
    signal          make_negative : std_logic ;
    signal          one_complement : std_logic ;
    signal          make_positive : std_logic ;
    signal          round_off : std_logic ;
    signal          two_complement_double : std_logic ;
    signal          two_complement : std_logic ;
```



```
    -- jump
    signal          jump : std_logic ;
    signal          jump_bootstrap : std_logic ;
    signal          jump_on_carry : std_logic ;
    signal          jump_equal : std_logic ;
    signal          jump_geater_equal : std_logic ;
    signal          jump_less : std_logic ;
    signal          jump_not_equal : std_logic ;
    signal          jump_over_flow : std_logic ;
    signal          jump_power_out : std_logic ;
    signal          jump_link_memory : std_logic ;
    signal          jump_link_register : std_logic ;
    signal          jump_negative : std_logic ;
    signal          jump_positive : std_logic ;
    signal          jump_not_zero : std_logic ;
    signal          jump_zero : std_logic  ;
    signal          zero : std_logic  ;
    signal          nzero : std_logic ;
    signal          pos : std_logic  ;
    signal          neg : std_logic  ;
    signal          carry : std_logic  ;
    signal          ovr : std_logic  ;

END pccon ;

ARCHITECTURE behaviour OF pccon  IS
--compass dp_gates;
BEGIN
    -- decoder operation
PROCESS (Inputdata, m, opdeco_en,
           byte_load ,
           byte_load_index ,
           byte_store_index ,
           double_load_index ,
           literal_load ,
           load_multiple ,
           load_mult_reg ,
           store_mult_reg ,
           word_load_index ,
           move_block ,
           load ,
           load_bit ,
           load_double ,
           load_mult ,
           byte_store ,
           store ,
           store_double ,
           store_address ,
           store_zero ,
           decindex_store ,
           decindex_store_double ,
           load_store ,
          jump_busy ,
          jump_input ,
         jump_io ,
         add ,
         add_byte ,
         add_double ,
         subtract ,
         subtract_byte ,
         subtract_double ,
         multiply ,
         multiply_double ,
         divide ,
```

```
divide_double ,
fpoint_add ,
fpoint_divide ,
fpoint_mult ,
fpoint_subtract ,
literal_add ,
literal_double ,
sign_ext_double ,
sign_ext ,
and_op ,
or_op ,
xor_op ,
byte_comp ,
comp ,
bit_comp ,
comp_double ,
logical_comp ,
masked_comp ,
literal_comp ,
float_comp ,
 alg_ldouble_shft ,
alg_left_shft ,
alg_right_shft ,
alg_rdouble_shft ,
cir_left_shft ,
cir_ldouble_shft ,
literal_left_shft ,
literal_ldouble_shft ,
literal_right_shft ,
literal_rdouble_shft ,
logical_right_shft ,
logical_rdouble_shft ,
  stack_get_top ,
  stack_put_top ,
  queue_get_top ,
  queue_put_top ,
  queue_put_bottom ,
   float_to_single ,
   float_to_double ,
   fixed_to_float ,
   fixed_double_to_float ,
   floating_normalize ,
   algebraic_left_quadruple_shift ,
   algebraic_right_quadruple_shift ,
   executive_return ,
   count_ones ,
   biased_fetch ,
   masked_substitute ,
   remote_execute ,
   set_bit ,
   zero_bit ,
   reverse_register ,
   scale_factor ,
       disable_clock_int ,
       disable_rtc_register ,
       disable_monitor_clock ,
       enable_clock_int ,
       enable_clock_and_int ,
       load_rtc_lower ,
       load_double_and_clock ,
       load_enable_monitor_clock ,
       load_clock ,
       reset_bit_timer ,
       diagnostic_jump ,
```



```
       jump_stop_key ,
       jump_stop_key2 ,
       jump_after_stop ,
       load_addr_register ,
       load_physical_addr ,
       load_physical_location ,
       store_physical_location ,
       load_sr1 ,
       load_sr2 ,
        set_bit_indicator ,
        init_processor_int ,
        ipl_failed ,
        load_p ,
        store_clock ,
        store_clock_double ,
        store_sr1 ,
        store_sr2 ,
        store_monitor_clock ,
        fixed_point_double_arccosine ,
        fixed_point_double_arcsine ,
        fixed_point_double_arctan ,
        fixed_point_double_exponent ,
        fixed_point_cross_product ,
        fixed_point_matrix_multiply ,
        fixed_point_double_natural_log ,
        fixed_point_sum_square ,
        fixed_point_double_sine_cosine ,
        fixed_point_double_square_root ,
        decrease_ra_1 ,
        decrease_ra_2 ,
        increase_ra_1 ,
        make_negative ,
        one_complement ,
        make_positive ,
        round_off ,
        two_complement_double ,
        two_complement ,
       jump ,
       jump_bootstrap ,
       jump_on_carry ,
       jump_equal ,
       jump_geater_equal ,
       jump_less ,
       jump_not_equal ,
       jump_over_flow ,
       jump_power_out ,
       jump_link_memory ,
       jump_link_register ,
       jump_negative ,
       jump_positive ,
       jump_not_zero ,
       jump_zero ,
       zero ,
       nzero ,
       pos ,
       neg ,
       carry ,
       ovr )

    variable op : bit_vector(5 downto 0);


BEGIN
```

```
-- $$$$$$$$$$$$$$$$$$  Begin decoding the op code format $$$$$$$$$$$$$$$$$$$$$$$$$$$

 if opdeco_en = '1' then
        op := to_bitvector(inputdata(15 downto 10));

    -- decode instruction set
       case op is
            -- LOAD/STORE
                 when op_byte_load(7 downto 2) => byte_load <= '1' ;

                 when OTHERS => byte_load <= '0'    ;

                 end case ;
       case op is
                 when op_byte_load_index(7 downto 2) => byte_load_index <= '1' ;

                 when OTHERS =>   byte_load_index <= '0'  ;
                 end case ;
       case op is
                 when op_byte_store_index(7 downto 2) => byte_store_index <= '1' ;

                 when OTHERS =>    byte_store_index <= '0' ;
                 end case ;
       case op is
                 when op_double_load_index(7 downto 2) => double_load_index <= '1' ;
                 when OTHERS =>  double_load_index <= '0'   ;
                 end case ;
       case op is
                 when op_literal_load(7 downto 2) => literal_load <= '1';
                 when OTHERS =>  literal_load <= '0'   ;
                 end case ;
       case op is
                 when op_load_multiple(7 downto 2) => load_multiple <= '1';
                 when OTHERS => load_multiple <= '0'     ;
                 end case ;
       case op is
                 when op_load_mult_reg(7 downto 2)  => load_mult_reg <= '1';
                 when OTHERS =>   load_mult_reg <= '0'  ;
                 end case ;
       case op is
                 when op_store_mult_reg(7 downto 2)  => store_mult_reg <= '1';
                 when OTHERS =>  store_mult_reg <= '0'   ;
                 end case ;
       case op is
                 when op_word_load_index(7 downto 2)  => word_load_index <= '1';
                 when OTHERS =>   word_load_index <= '0'   ;
                 end case ;
       case op is
                 when op_move_block(7 downto 2) => move_block <= '1';
                 when OTHERS => move_block <= '0'    ;
                 end case ;
       case op is
                 when op_load(7 downto 2) => load <= '1' ;
                 when OTHERS =>  load <= '0'    ;
                 end case ;
       case op is
                 when op_load_bit(7 downto 2) => load_bit <= '1';
                 when OTHERS =>  load_bit <= '0'   ;
                 end case ;
       case op is
                 when op_load_double(7 downto 2) => load_double <= '1';
                 when OTHERS => load_double <= '0'    ;
                 end case ;
```



```
       case op is
                 when op_load_mult(7 downto 2) => load_mult <= '1';
                 when OTHERS => load_mult <= '0'    ;
                 end case ;
       case op is
                 when op_byte_store(7 downto 2) => byte_store <= '1' ;
                 when OTHERS =>  byte_store <= '0'    ;
                 end case ;
       case op is
                 when op_store(7 downto 2) => store <= '1' ;
                 when OTHERS =>   store <= '0'    ;
                 end case ;
       case op is
                 when op_store_double(7 downto 2) => store_double <= '1';
                 when OTHERS => store_double <= '0'    ;
                 end case ;
       case op is
                 when op_store_address(7 downto 2) => store_address <= '1';
                 when OTHERS =>   store_address <= '0'   ;
                 end case ;
       case op is
                 when op_store_zero(7 downto 2) => store_zero <= '1';
                 when OTHERS =>  store_zero <= '0'   ;
                 end case ;
       case op is
                 when op_decindex_store(7 downto 2) => decindex_store <= '1';
                 when OTHERS =>  decindex_store <= '0'    ;
                 end case ;
       case op is
                 when op_decindex_store_double(7 downto 2) => decindex_store_double <= '1';
                 when OTHERS => decindex_store_double <= '0'    ;
                 end case ;


       case op is
                 when op_jump_io(7 downto 2) => jump_io <= '1';
                 when OTHERS => jump_io <= '0'    ;
                 end case ;
       case op is
                 when op_jump_link_memory(7 downto 2) => jump_link_memory <= '1';
                 when OTHERS =>   jump_link_memory <= '0'  ;
                 end case ;
       case op is
                 when op_jump_link_register(7 downto 2) => jump_link_register <= '1';
                 when OTHERS => jump_link_register <= '0'    ;
                 end case ;
       case op is
                 when op_jump_negative(7 downto 2) => jump_negative <= '1';
                 when OTHERS => jump_negative <= '0'    ;
                 end case ;
       case op is
                 when op_jump_not_zero(7 downto 2) => jump_not_zero <= '1';
                 when OTHERS =>jump_not_zero <= '0'     ;
                 end case ;
       case op is
                 when op_jump_positive(7 downto 2) => jump_positive <= '1';
                 when OTHERS =>  jump_positive <= '0'   ;
                 end case ;
       case op is
                 when op_jump_zero(7 downto 2) => jump_zero <= '1';
                 when OTHERS => jump_zero <= '0'    ;
                 end case ;

       -- ARITHMETIC
       case op is
```

```
            when op_add(7 downto 2) => add <= '1';
            when OTHERS =>   add <= '0'   ;
            end case ;
case op is
            when op_add_byte(7 downto 2) => add_byte <= '1';
            when OTHERS => add_byte <= '0'    ;
            end case ;
case op is
            when op_add_double(7 downto 2) => add_double <= '1';
            when OTHERS =>  add_double <= '0'    ;
            end case ;
case op is
            when op_subtract(7 downto 2) => subtract <= '1';
            when OTHERS =>   subtract <= '0'  ;
            end case ;
case op is
            when op_subtract_byte(7 downto 2) => subtract_byte <= '1';
            when OTHERS => subtract_byte <= '0'    ;
            end case ;
case op is
            when op_subtract_double(7 downto 2) => subtract_double <= '1';
            when OTHERS =>  subtract_double <= '0'   ;
            end case ;
case op is
            when op_multiply(7 downto 2) => multiply <= '1';
            when OTHERS => multiply <= '0'     ;
            end case ;
case op is
            when op_multiply_double(7 downto 2) => multiply_double <= '1';
            when OTHERS => multiply_double <= '0'    ;
            end case ;
case op is
            when op_divide(7 downto 2) => divide <= '1';
            when OTHERS => divide <= '0';
            end case ;
case op is
            when op_divide_double(7 downto 2) => divide_double <= '1';
            when OTHERS =>  divide_double <= '0'   ;
            end case ;
case op is
            when op_fpoint_add(7 downto 2) => fpoint_add <= '1';
            when OTHERS =>  fpoint_add <= '0'    ;
            end case ;
case op is
            when op_fpoint_divide(7 downto 2) => fpoint_divide <= '1';
            when OTHERS =>  fpoint_divide <= '0'    ;
            end case ;
case op is
            when op_fpoint_mult(7 downto 2) => fpoint_mult <= '1';
            when OTHERS => fpoint_mult <= '0'     ;
            end case ;
case op is
            when op_fpoint_subtract(7 downto 2) => fpoint_subtract <= '1';
            when OTHERS => fpoint_subtract <= '0'    ;
            end case ;
case op is
            when op_literal_add(7 downto 2) => literal_add <= '1';
            when OTHERS => literal_add <= '0'    ;
            end case ;
case op is
            when op_literal_double(7 downto 2) => literal_double <= '1';
            when OTHERS => literal_double <= '0'    ;
            end case ;
case op is
```



```
            when op_sign_ext_double(7 downto 2) => sign_ext_double <= '1';
            when OTHERS =>  sign_ext_double <= '0'   ;
            end case ;
case op is
            when op_sign_ext(7 downto 2) => sign_ext <= '1';
            when OTHERS => sign_ext <= '0'    ;
            end case ;

 -- LOGICAL
case op is
            when op_and(7 downto 2) => and_op <= '1';
            when OTHERS => and_op <= '0'    ;
            end case ;
case op is
            when op_or(7 downto 2) => or_op <= '1';
            when OTHERS => or_op <= '0'    ;
            end case ;
case op is
            when op_xor(7 downto 2) => xor_op <= '1';
            when OTHERS => xor_op <= '0'     ;
            end case ;

 -- COMPARE
case op is
            when op_byte_comp(7 downto 2) => byte_comp <= '1';
            when OTHERS =>  byte_comp <= '0'    ;
            end case ;
case op is
            when op_comp(7 downto 2) => comp <= '1';
            when OTHERS => comp <= '0'    ;
            end case ;
case op is
            when op_bit_comp(7 downto 2) => bit_comp <= '1';
            when OTHERS => bit_comp <= '0'    ;
            end case ;
case op is
            when op_comp_double(7 downto 2) => comp_double <= '1';
            when OTHERS => comp_double <= '0'    ;
            end case ;
case op is
            when op_logical_comp(7 downto 2) => logical_comp <= '1';
            when OTHERS => logical_comp <= '0'    ;
            end case ;
case op is
            when op_masked_comp(7 downto 2) => masked_comp <= '1';
            when OTHERS => masked_comp <= '0'    ;
            end case ;
case op is
            when op_literal_comp(7 downto 2) => literal_comp <= '1';
            when OTHERS => literal_comp <= '0'    ;
            end case ;

 -- SHIFT
case op is
            when op_alg_ldouble_shft(7 downto 2) => alg_ldouble_shft <= '1';
            when OTHERS =>  alg_ldouble_shft <= '0'    ;
            end case ;
case op is
            when op_alg_left_shft(7 downto 2) => alg_left_shft <= '1';
            when OTHERS => alg_left_shft <= '0'    ;
            end case ;
case op is
            when op_alg_right_shft(7 downto 2) => alg_right_shft <= '1';
            when OTHERS => alg_right_shft <= '0'    ;
```

```
            end case ;
      case op is
            when op_alg_rdouble_shft(7 downto 2) => alg_rdouble_shft <= '1';
            when OTHERS => alg_rdouble_shft <= '0'    ;
            end case ;
      case op is
            when op_cir_left_shft(7 downto 2) => cir_left_shft <= '1';
            when OTHERS =>  cir_left_shft <= '0'   ;
            end case ;

      case op is
            when op_cir_ldouble_shft(7 downto 2) => cir_ldouble_shft <= '1';
            when OTHERS =>  cir_ldouble_shft <= '0'    ;
            end case ;
      case op is
            when op_literal_left_shft(7 downto 2) => literal_left_shft <= '1';
            when OTHERS => literal_left_shft <= '0'    ;
            end case ;
      case op is
            when op_literal_ldouble_shft(7 downto 2) => literal_ldouble_shft <= '1';
            when OTHERS =>  literal_ldouble_shft <= '0'   ;
            end case ;
      case op is
            when op_literal_right_shft(7 downto 2) => literal_right_shft <= '1';
            when OTHERS =>  literal_right_shft <= '0'   ;
            end case ;
      case op is
            when op_literal_rdouble_shft(7 downto 2) => literal_rdouble_shft <= '1';
            when OTHERS => literal_rdouble_shft <= '0'    ;
            end case ;
      case op is
            when op_logical_right_shft(7 downto 2) => logical_right_shft <= '1';
            when OTHERS =>  logical_right_shft <= '0'   ;
            end case ;
      case op is
            when op_logical_rdouble_shft(7 downto 2) => logical_rdouble_shft <= '1';
            when OTHERS =>logical_rdouble_shft <= '0'     ;
            end case ;


        -- STACK and QUEUE
      case op is
            when op_stack_get_top(7 downto 2)  => stack_get_top <= '1';
            when OTHERS =>  stack_get_top <= '0'    ;
            end case ;
      case op is
            when op_stack_put_top(7 downto 2)  => stack_put_top <= '1';
            when OTHERS => stack_put_top <= '0'    ;
            end case ;
      case op is
            when op_queue_get_top(7 downto 2)  => queue_get_top <= '1';
            when OTHERS =>  queue_get_top <= '0'   ;
            end case ;
      case op is
            when op_queue_put_top(7 downto 2)  => queue_put_top <= '1';
            when OTHERS => queue_put_top <= '0'    ;
            end case ;
      case op is
            when op_queue_put_bottom(7 downto 2) => queue_put_bottom <= '1';
            when OTHERS => queue_put_bottom <= '0'    ;
            end case ;

         -- MISCELLANEOUS
      case op is
```



```
            when op_biased_fetch(7 downto 2) => biased_fetch <= '1';
            when OTHERS => biased_fetch <= '0'    ;
            end case ;

      case op is
            when op_masked_substitute(7 downto 2) => masked_substitute <= '1';
            when OTHERS => masked_substitute <= '0'    ;
            end case ;
      case op is
            when op_remote_execute(7 downto 2) => remote_execute <= '1';
            when OTHERS =>  remote_execute <= '0'   ;
            end case ;
      case op is
            when op_set_bit(7 downto 2) => set_bit <= '1';
            when OTHERS => set_bit <= '0'    ;
            end case ;
      case op is
            when op_zero_bit(7 downto 2) => zero_bit <= '1';
            when OTHERS => zero_bit <= '0'    ;
            end case ;
         -- EXECUTIVE MODE
      case op is
            when  op_load_addr_register(7 downto 2) => load_addr_register <= '1';
            when OTHERS =>load_addr_register <= '0'     ;
            end case ;
      case op is
            when  op_load_physical_addr(7 downto 2) => load_physical_addr <= '1';
            when OTHERS => load_physical_addr <= '0'    ;
            end case ;
      case op is
            when  op_load_physical_location(7 downto 2) => load_physical_location <= '1'
            when OTHERS => load_physical_location <= '0'    ;
            end case ;
      case op is
            when  op_store_physical_location(7 downto 2) => store_physical_location <= '
          when OTHERS =>  store_physical_location <= '0'   ;
          end case ;


    case op is
--     if op = op_load_store(7 downto 2) then
         when op_load_store(7 downto 2) =>
                    case m is
                        when "0100" => load_p <= '1'; --4
                        when others => load_p <= '0' ;
                    end case ;

     when others => load_p <= '0' ;
    end case ;

    case op is
          when op_load_store(7 downto 2) =>
                    case m is
                        when "0011" => store_clock <= '1'; --3
                        when others =>store_clock <= '0' ;
                    end case ;
                       when others =>store_clock <= '0' ;
     end case ;


     case op is
          when op_load_store(7 downto 2) =>
                    case m is
                        when "1101" => store_clock_double <= '1'; --D
```

```
                    when others => store_clock_double <= '0' ;
                end case ;
                    when others => store_clock_double <= '0' ;
                end case ;

case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "0001" => store_sr1 <= '1'; --1
                    when others => store_sr1 <= '0';
                end case ;
                   when others => store_sr1 <= '0';
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "0010" => store_sr2 <= '1'; --2
                    when others => store_sr2 <= '0' ;
                end case ;
                    when others => store_sr2 <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "0000" => executive_return <= '1'; --0
                    when others => executive_return <= '0' ;
                end case ;
                   when others => executive_return <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "1111" => disable_clock_int <= '1'; --F
                    when others => disable_clock_int <= '0' ;
                end case ;
                   when others => disable_clock_int <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "1001" => disable_rtc_register <= '1'; --9
                    when others => disable_rtc_register <= '0' ;
                end case ;
                    when others => disable_rtc_register <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "1011" => disable_monitor_clock <= '1'; --B
                    when others =>disable_monitor_clock <= '0' ;
                end case ;
                    when others =>disable_monitor_clock <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
               case m is
                    when "1110" => enable_clock_int  <= '1'; --E
                    when others =>enable_clock_int  <= '0' ;
               end case ;

                    when others =>enable_clock_int  <= '0' ;
               end case ;
case op is
      when op_load_store(7 downto 2) =>
```



```
                case m is
                    when "1000" => enable_clock_and_int <= '1'; --8
                    when others => enable_clock_and_int <= '0' ;
                end case ;
                    when others => enable_clock_and_int <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "0111" => load_rtc_lower <= '1'; --7
                    when others =>  load_rtc_lower <= '0';
                end case ;
                    when others =>  load_rtc_lower <= '0';
                end case ;

case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "1100" => load_double_and_clock <= '1'; --C
                    when others => load_double_and_clock <= '0';
                end case ;
                   when others => load_double_and_clock <= '0';
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "1010" => load_enable_monitor_clock <= '1'; --A
                    when others => load_enable_monitor_clock <= '0' ;
                end case ;
                   when others => load_enable_monitor_clock <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "0101" => load_sr1 <= '1'; --5
                    when others => load_sr1 <= '0' ;
                end case ;
                    when others => load_sr1 <= '0' ;
                end case ;
case op is
      when op_load_store(7 downto 2) =>
                case m is
                    when "0110" => load_sr2 <= '1'; --6
                    when others => load_sr2 <= '0' ;
                end case ;

                when others => load_sr2 <= '0' ;
   end case;
--         end if ;


--              if op = op_store_monitor_clock(7 downto 2) then
case op is
      when op_store_monitor_clock(7 downto 2) =>
                   case m is
                     when "0100" => store_monitor_clock <= '1'; --4
                     when others => store_monitor_clock <= '0' ;
                   end case ;
                    when others => store_monitor_clock <= '0' ;
                   end case ;
case op is
      when op_store_monitor_clock(7 downto 2) =>
                   case m is
                     when "0010" => count_ones <= '1'; --2
```

```
                        when others => count_ones <= '0' ;
                      end case ;
                        when others => count_ones <= '0' ;
                      end case ;
      case op is
            when op_store_monitor_clock(7 downto 2) =>
                      case m is
                        when "0001" => reverse_register <= '1'; --1
                        when others => reverse_register <= '0' ;
                      end case ;
                        when others => reverse_register <= '0' ;
                      end case ;
      case op is
            when op_store_monitor_clock(7 downto 2) =>
                      case m is
                        when "0011" => scale_factor <= '1'; --3
                        when others => scale_factor <= '0';
                      end case ;
                        when others => scale_factor <= '0';
                      end case ;
      case op is
            when op_store_monitor_clock(7 downto 2) =>
                      case m is
                        when "0110" => load_clock   <= '1'; --6
                        when others =>  load_clock   <= '0' ;
                      end case ;

              when others => load_clock <= '0';
        end case ;
--              end if ;

       -- SUPPORT CHANNEL / JUMP

--      if op = op_jump(7 downto 2) then
      case op is
        when op_jump(7 downto 2) =>
             case a is
                        when "1100" => jump_busy <= '1'; -- C
                        when others => jump_busy <= '0' ;
                end case ;
                      when others => jump_busy <= '0' ;
                end case ;
      case op is
        when op_jump(7 downto 2) =>
                case a is
                        when "1000" => jump <= '1'; -- 8
                        when others => jump <= '0' ;
                end case ;
                        when others => jump <= '0' ;
                end case ;
      case op is
        when op_jump(7 downto 2) =>
                case a is
                        when "0111" => jump_bootstrap <= '1'; -- 7
                        when others => jump_bootstrap <= '0' ;
                end case ;
                        when others => jump_bootstrap <= '0' ;
                end case ;
      case op is
        when op_jump(7 downto 2) =>
                case a is
                        when "0101" => jump_on_carry <= '1'; --5
                        when others => jump_on_carry <= '0' ;
                end case ;
```



```
                        when others => jump_on_carry <= '0' ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "0000" => jump_equal <= '1'; --0
                        when others => jump_equal <= '0' ;
                end case ;
                      when others => jump_equal <= '0' ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "0010" => jump_geater_equal <= '1'; --2
                        when others => jump_geater_equal <= '0' ;
                end case ;
                     when others => jump_geater_equal <= '0' ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "0011" => jump_less <= '1'; --3
                        when others => jump_less <= '0'  ;
                end case ;
                      when others => jump_less <= '0'  ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "0001" => jump_not_equal <= '1'; --1
                        when others =>jump_not_equal <= '0'  ;
                end case ;
                        when others =>jump_not_equal <= '0'  ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "0100" => jump_over_flow <= '1'; --4
                        when others => jump_over_flow <= '0' ;
                end case ;
                      when others => jump_over_flow <= '0' ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "0110" => jump_power_out <= '1'; --6
                        when others => jump_power_out <= '0' ;
                end case ;
                     when others => jump_power_out <= '0' ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "1010" => jump_stop_key <= '1'; --A
                        when others => jump_stop_key <= '0' ;
                end case ;
                     when others => jump_stop_key <= '0' ;
                end case ;
case op is
  when op_jump(7 downto 2) =>
                case a is
                        when "1011" => jump_stop_key2 <= '1'; --B
                        when others =>  jump_stop_key2 <= '0' ;
                end case ;
```

```
                    when others =>  jump_stop_key2 <= '0' ;
                end case ;
        case op is
          when op_jump(7 downto 2) =>
                case a is
                        when "1001" => jump_after_stop <= '1'; --9
                        when others => jump_after_stop <= '0' ;
                end case ;
              when others => jump_after_stop <= '0';

        end case ;
--          end if ;


--              if op = op_jump_input(7 downto 2) then
        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                        when "0111" => jump_input <= '1'; --7
                        when others =>jump_input <= '0'  ;
                      end case ;
                        when others =>jump_input <= '0'  ;
                      end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "1001" => decrease_ra_1 <= '1'; --9
                       when others => decrease_ra_1 <= '0' ;
                    end case ;
                      when others => decrease_ra_1 <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "1011" => decrease_ra_2 <= '1'; --B
                       when others => decrease_ra_2 <= '0' ;
                    end case ;
                      when others => decrease_ra_2 <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "1000" => increase_ra_1 <= '1'; --8
                       when others => increase_ra_1 <= '0' ;
                    end case ;
                      when others => increase_ra_1 <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "0001" => make_negative <= '1'; --1
                       when others =>  make_negative <= '0';
                    end case ;
                     when others =>  make_negative <= '0';
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "0110" => one_complement <= '1'; --6
                       when others =>  one_complement <= '0' ;
                    end case ;
                      when others =>  one_complement <= '0' ;
                    end case ;

        case op is
```



```
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "0000" => make_positive <= '1'; --0
                       when others =>  make_positive <= '0' ;
                    end case ;
                      when others =>  make_positive <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "0010" => round_off      <= '1'; --2
                       when others =>  round_off      <= '0' ;
                    end case ;
                       when others =>  round_off      <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "0101" => two_complement_double <= '1'; --5
                       when others => two_complement_double <= '0' ;
                    end case ;
                    when others => two_complement_double <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "0100" => two_complement <= '1'; --4
                       when others => two_complement <= '0' ;
                    end case ;
                       when others => two_complement <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "1110" => reset_bit_timer <= '1'; --E
                       when others => reset_bit_timer <= '0' ;
                    end case ;
                     when others => reset_bit_timer <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "1101" => diagnostic_jump <= '1'; --D
                       when others => diagnostic_jump <= '0' ;
                    end case ;
                       when others => diagnostic_jump <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "1111" => set_bit_indicator <= '1'; --F
                       when others => set_bit_indicator <= '0' ;
                    end case ;
                       when others => set_bit_indicator <= '0' ;
                    end case ;

        case op is
          when op_jump_input(7 downto 2) =>
                    case m is
                       when "0011" => init_processor_int <= '1'; --3
                       when others => init_processor_int <= '0' ;
                    end case ;
                     when others => init_processor_int <= '0' ;
                    end case ;

        case op is
```

```
            when op_jump_input(7 downto 2) =>
                        case m is
                           when "1100" => ipl_failed <= '1'; --C
                           when others => ipl_failed <= '0' ;
                        end case ;
             when others => ipl_failed <= '0' ;
             end case ;
--                end if ;


--        if op = op_floating_comp(7 downto 2) then
        case op is
           when op_floating_comp(7 downto 2) =>
                    case m is
                         when "1000" => float_comp <= '1';
                         when others =>float_comp <= '0'  ;
                      end case ;
                       when others =>float_comp <= '0'  ;
                      end case ;
        case op is
           when op_floating_comp(7 downto 2) =>
                      case m is
                          when "1010" => float_to_single <= '1'; --A
                          when others =>float_to_single <= '0'  ;
                      end case ;
                       when others =>float_to_single <= '0'  ;
                      end case ;
        case op is
           when op_floating_comp(7 downto 2) =>
                      case m is
                          when "1101" => float_to_double <= '1'; --D
                          when others =>float_to_double <= '0'  ;
                      end case ;
                         when others =>float_to_double <= '0'  ;
                      end case ;
        case op is
           when op_floating_comp(7 downto 2) =>
                      case m is
                          when "1001" => fixed_to_float <= '1'; --9
                          when others => fixed_to_float <= '0' ;
                      end case ;
                         when others => fixed_to_float <= '0' ;
                      end case ;
        case op is
           when op_floating_comp(7 downto 2) =>
                      case m is
                          when "1100" => fixed_double_to_float <= '1'; --C
                          when others => fixed_double_to_float <= '0' ;
                      end case ;
                      when others => fixed_double_to_float <= '0' ;
                      end case ;
        case op is
           when op_floating_comp(7 downto 2) =>
                      case m is
                          when "1011" => floating_normalize <= '1'; --B
                          when others => floating_normalize <= '0' ;
                      end case ;
                        when others => floating_normalize <= '0' ;
                      end case ;
        case op is
           when op_floating_comp(7 downto 2) =>
                      case m is
                          when "1110" => algebraic_left_quadruple_shift <= '1'; --E
                          when others => algebraic_left_quadruple_shift <= '0' ;
```



```
                      end case ;
                         when others => algebraic_left_quadruple_shift <= '0' ;
                      end case ;
        case op is
           when op_floating_comp(7 downto 2) =>
                      case m is
                          when "1111" => algebraic_right_quadruple_shift <= '1'; --F
                          when others =>algebraic_right_quadruple_shift <= '0'  ;
                      end case ;

         when others =>algebraic_right_quadruple_shift <= '0'  ;
         end case ;
--                end if ;

          case op is
           when op_fixed_point(7 downto 2) =>
             --    if op = op_fixed_point(7 downto 2) then
                      case m is
                         when "0011" => fixed_point_double_arccosine    <= '1';
                         when others => fixed_point_double_arccosine    <= '0' ;
                      end case ;
                        when others => fixed_point_double_arccosine    <= '0' ;
                      end case ;
          case op is
           when op_fixed_point(7 downto 2) =>
                      case m is
                         when "0010" => fixed_point_double_arcsine      <= '1';
                         when others => fixed_point_double_arcsine      <= '0' ;
                      end case ;
                        when others => fixed_point_double_arcsine      <= '0' ;
                      end case ;
          case op is
           when op_fixed_point(7 downto 2) =>
                      case m is
                         when "0100" => fixed_point_double_arctan       <= '1';
                         when others => fixed_point_double_arctan       <= '0' ;
                      end case ;
                        when others => fixed_point_double_arctan       <= '0' ;
                      end case ;
          case op is
           when op_fixed_point(7 downto 2) =>
                      case m is
                         when "0111" => fixed_point_cross_product       <= '1';
                         when others => fixed_point_cross_product       <= '0' ;
                      end case ;
                         when others => fixed_point_cross_product       <= '0' ;
                      end case ;
          case op is
           when op_fixed_point(7 downto 2) =>
                      case m is
                         when "0001" => fixed_point_double_exponent     <= '1';
                         when others => fixed_point_double_exponent     <= '0' ;
                      end case ;
                        when others => fixed_point_double_exponent     <= '0' ;
                      end case ;
          case op is
           when op_fixed_point(7 downto 2) =>
                      case m is
                         when "1001" => fixed_point_matrix_multiply     <= '1';
                         when others => fixed_point_matrix_multiply     <= '0' ;
                      end case ;
                        when others => fixed_point_matrix_multiply     <= '0' ;
                      end case ;
          case op is
```

```
            when op_fixed_point(7 downto 2) =>
                        case m is
                           when "0110" => fixed_point_double_natural_log  <= '1';
                           when others =>fixed_point_double_natural_log  <= '0'  ;
                        end case ;
                          when others =>fixed_point_double_natural_log  <= '0'  ;
                        end case ;
           case op is
            when op_fixed_point(7 downto 2) =>
                        case m is
                           when "1000" => fixed_point_sum_square          <= '1';
                           when others =>   fixed_point_sum_square        <= '0';
                        end case ;
                          when others =>   fixed_point_sum_square        <= '0';
                        end case ;
           case op is
            when op_fixed_point(7 downto 2) =>
                        case m is
                           when "0000" => fixed_point_double_sine_cosine  <= '1';
                           when others => fixed_point_double_sine_cosine  <= '0' ;
                        end case ;
                           when others => fixed_point_double_sine_cosine  <= '0' ;
                        end case ;
           case op is
            when op_fixed_point(7 downto 2) =>
                        case m is
                           when "0101" => fixed_point_double_square_root  <= '1';
                           when others =>  fixed_point_double_square_root  <= '0' ;
                        end case ;
              when others =>  fixed_point_double_square_root  <= '0' ;
         end case ;


-- ***********  Done decode for instruction code *************
-- $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

 end if ; -- opdeco_en = '1'

sins <=  byte_load or byte_load_index or byte_store_index or
         literal_load or load or load_bit or  byte_store or
         store or store_address or store_zero or decindex_store or
         load_store or add or
             add_byte or
             subtract or
             subtract_byte or
             multiply or
             divide or
             fpoint_add or
             fpoint_divide or
             fpoint_mult or
             fpoint_subtract or
             literal_add or
             sign_ext or
             and_op or
             or_op or
             xor_op or
             byte_comp or
             comp or
             bit_comp or
             logical_comp or
             masked_comp or
             literal_comp or
             float_comp or
             alg_left_shft or
```



```
             alg_right_shft or
             cir_left_shft or
             literal_left_shft or
             literal_right_shft or
             logical_right_shft or
               stack_get_top or
               stack_put_top or
               queue_get_top or
               queue_put_top or
               queue_put_bottom or
                float_to_single or
                float_to_double or
                fixed_to_float or
                fixed_double_to_float or
                floating_normalize or
                algebraic_left_quadruple_shift or
                algebraic_right_quadruple_shift or
                executive_return or
                count_ones or
                biased_fetch or
                masked_substitute or
                remote_execute or
                set_bit or
                zero_bit or
                reverse_register or
                scale_factor or
                    disable_clock_int or
                    disable_rtc_register or
                    disable_monitor_clock or
                    enable_clock_int or
                    enable_clock_and_int or
                    load_rtc_lower or
                    load_double_and_clock or
                    load_enable_monitor_clock or
                    load_clock or
                    reset_bit_timer or
                    diagnostic_jump or
                   load_addr_register or
                   load_physical_addr or
                   load_physical_location or
                   store_physical_location or
                   load_sr1 or
                   load_sr2 or
                    set_bit_indicator or
                    init_processor_int or
                    ipl_failed or
                    load_p or
                    store_clock or
                    store_sr1 or
                    store_sr2 or
                    store_monitor_clock or
                    fixed_point_cross_product or
                    fixed_point_matrix_multiply or
                    fixed_point_sum_square or
                    decrease_ra_1 or
                    decrease_ra_2 or
                    increase_ra_1 or
                    make_negative or
                    one_complement or
                    make_positive or
                    round_off or
                    two_complement ;

dins <=  double_load_index or load_double or load_multiple or
```

```
        load_mult_reg or store_mult_reg or word_load_index or
        move_block or load_mult or store_double or
            decindex_store_double or
            add_double or
            subtract_double or
            multiply_double or
            divide_double or
            literal_double or
            sign_ext_double or
            comp_double or
             alg_ldouble_shft or
            alg_rdouble_shft or
            cir_ldouble_shft or
            literal_ldouble_shft or
            literal_rdouble_shft or
            logical_rdouble_shft or
            store_clock_double or
            fixed_point_double_arccosine or
            fixed_point_double_arcsine or
            fixed_point_double_arctan or
            fixed_point_double_exponent or
            fixed_point_double_natural_log or
            fixed_point_double_sine_cosine or
            fixed_point_double_square_root or
            two_complement_double;

   bins <= jump or jump_bootstrap or
           jump_power_out or jump_link_memory or jump_link_register or
            jump_stop_key or jump_stop_key2 or jump_after_stop;

    selmr <=  byte_load  or
           byte_load_index  or
           byte_store_index  or
           double_load_index  or
           literal_load  or
           load_multiple  or
           load_mult_reg  or
           load  or
           load_bit  or
           load_double  or
           load_mult  or
               load_addr_register  or
               load_physical_addr  or
               load_physical_location ;
sr1op  <= load_sr1;
sr2op  <= load_sr2;
fpins  <=  fpoint_add  or
           fpoint_divide  or
           fpoint_mult  or
           fpoint_subtract;
andop  <=  and_op ;
orop   <=  or_op ;
xorop  <=  xor_op ;
END PROCESS;
END behaviour ;
```

```
-- ***************************************************
-- This vhdl code readin 9 bit input and decode
-- to 300 output bit.
-- ***************************************************

--USE std.std_logic.ALL;
--USE std.std.ttl.ALL;
library IEEE;
use IEEE.std_logic_1164.all;

library COMPASS_LIB;
use COMPASS_LIB.COMPASS.all;
--compass compile_off
use COMPASS_LIB.COMPASS_ETC.ALL;
--compass compile_on

use Work.Emulator_datatype.all ;

ENTITY romdeco IS
   PORT (rominput  : IN std_logic_vector(8 downto 0);
         rom_addr  : OUT std_logic_vector(299 downto 0)) ;

END romdeco ;

ARCHITECTURE behaviour OF romdeco IS
--compass dp_gates;
BEGIN
    -- decoder operation
PROCESS (rominput)

       variable op : bit_vector(5 downto 0);
       variable b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15,b16,b17,b18,b19,
                b20,b21,b22,b23,b24,b25,b26,b27,b28,b29,b30,b31,b32,b33,b34,b35,b36,b37,b38,b39
                b40,b41,b42,b43,b44,b45,b46,b47,b48,b49,b50,b51,b52,b53,b54,b55,b56,b57,b58,b59
                b60,b61,b62,b63,b64,b65,b66,b67,b68,b69,b70,b71,b72,b73,b74,b75,b76,b77,b78,b79
                b80,b81,b82,b83,b84,b85,b86,b87,b88,b89,b90,b91,b92,b93,b94,b95,b96,b97,b98,b99

                b100,b101,b102,b103,b104,b105,b106,b107,b108,b109,b110,b111,b112,b113,b114,b115
                b120,b121,b122,b123,b124,b125,b126,b127,b128,b129,b130,b131,b132,b133,b134,b135
                b140,b141,b142,b143,b144,b145,b146,b147,b148,b149,b150,b151,b152,b153,b154,b155
                b160,b161,b162,b163,b164,b165,b166,b167,b168,b169,b170,b171,b172,b173,b174,b175
                b180,b181,b182,b183,b184,b185,b186,b187,b188,b189,b190,b191,b192,b193,b194,b195

                b200,b201,b202,b203,b204,b205,b206,b207,b208,b209,b210,b211,b212,b213,b214,b215
                b220,b221,b222,b223,b224,b225,b226,b227,b228,b229,b230,b231,b232,b233,b234,b235
                b240,b241,b242,b243,b244,b245,b246,b247,b248,b249,b250,b251,b252,b253,b254,b255
                b260,b261,b262,b263,b264,b265,b266,b267,b268,b269,b270,b271,b272,b273,b274,b275
            b280,b281,b282,b283,b284,b285,b286,b287,b288,b289,b290,b291,b292,b293,b294,b295,b29

  BEGIN


 -- $$$$$$$$$$$$$$$$$$  Begin decoding the op code format $$$$$$$$$$$$$$$$$$$$$$$$$

--  if opdeco_en = '1' then
  --        op := to_bitvector(inputdata(15 downto 10));

     -- decode instruction set
        case rominput is
                  when "000000000" => b0 := '1';
                  when OTHERS => b0 := '0';
        end case ;
        case rominput is
```



```
                  when "000000001" => b1 := '1';
                  when OTHERS => b1 := '0';
        end case ;
        case rominput is
                  when "000000010" => b2 := '1';
                  when OTHERS => b2 := '0';
        end case ;
        case rominput is
                  when "000000011" => b3 := '1';
                  when OTHERS => b3 := '0';
        end case ;
        case rominput is
                  when "000000100" => b4 := '1';
                  when OTHERS => b4 := '0';
        end case ;
        case rominput is
                  when "000000101" => b5 := '1';
                  when OTHERS => b5 := '0';
        end case ;
        case rominput is
                  when "000000110" => b6 := '1';
                  when OTHERS => b6 := '0';
        end case ;
        case rominput is
                  when "000000111" => b7 := '1';
                  when OTHERS => b7 := '0';
        end case ;
        case rominput is
                  when "000001000" => b8 := '1';
                  when OTHERS => b8 := '0';
        end case ;
        case rominput is
                  when "000001001" => b9 := '1';
                  when OTHERS => b9 := '0';
        end case ;  -- 9 case

        case rominput is
                  when "000001010" => b10 := '1';
                  when OTHERS => b10 := '0';
        end case ;
        case rominput is
                  when "000001011" => b11 := '1';
                  when OTHERS => b11 := '0';
        end case ;
        case rominput is
                  when "000001100" => b12 := '1';
                  when OTHERS => b12 := '0';
        end case ;
        case rominput is
                  when "000001101" => b13 := '1';
                  when OTHERS => b13 := '0';
        end case ;
        case rominput is
                  when "000001110" => b14 := '1';
                  when OTHERS => b14 := '0';
        end case ;
        case rominput is
                  when "000001111" => b15 := '1';
                  when OTHERS => b15 := '0';
        end case ;
        case rominput is
                  when "000010000" => b16 := '1';
                  when OTHERS => b16 := '0';
        end case ;
```

```
case rominput is
          when "000010001" => b17 := '1';
          when OTHERS => b17 := '0';
end case ;
case rominput is
          when "000010010" => b18 := '1';
          when OTHERS => b18 := '0';
end case ;
case rominput is
          when "000010011" => b19 := '1';
          when OTHERS => b19 := '0';
end case ;  -- 19 case

case rominput is
          when "000010100" => b20 := '1';
          when OTHERS => b20 := '0';
end case ;
case rominput is
          when "000010101" => b21 := '1';
          when OTHERS => b21 := '0';
end case ;
case rominput is
          when "000010110" => b22 := '1';
          when OTHERS => b22 := '0';
end case ;
case rominput is
          when "000010111" => b23 := '1';
          when OTHERS => b23 := '0';
end case ;
case rominput is
          when "000011000" => b24 := '1';
          when OTHERS => b24 := '0';
end case ;
case rominput is
          when "000011001" => b25 := '1';
          when OTHERS => b25 := '0';
end case ;
case rominput is
          when "000011010" => b26 := '1';
          when OTHERS => b26 := '0';
end case ;
case rominput is
          when "000011011" => b27 := '1';
          when OTHERS => b27 := '0';
end case ;
case rominput is
          when "000011100" => b28 := '1';
          when OTHERS => b28 := '0';
end case ;
case rominput is
          when "000011101" => b29 := '1';
          when OTHERS => b29 := '0';
end case ; -- 29 case

case rominput is
          when "000011110" => b30 := '1';
          when OTHERS => b30 := '0';
end case ;
case rominput is
          when "000011111" => b31 := '1';
          when OTHERS => b31 := '0';
end case ;
case rominput is
          when "000100000" => b32 := '1';
```



```
          when OTHERS => b32 := '0';
end case ;
case rominput is
          when "000100001" => b33 := '1';
          when OTHERS => b33 := '0';
end case ;
case rominput is
          when "000100010" => b34 := '1';
          when OTHERS => b34 := '0';
end case ;
case rominput is
          when "000100011" => b35 := '1';
          when OTHERS => b35 := '0';
end case ;
case rominput is
          when "000100100" => b36 := '1';
          when OTHERS => b36 := '0';
end case ;
case rominput is
          when "000100101" => b37 := '1';
          when OTHERS => b37 := '0';
end case ;
case rominput is
          when "000100110" => b38 := '1';
          when OTHERS => b38 := '0';
end case ;
case rominput is
          when "000100111" => b39 := '1';
          when OTHERS => b39 := '0';
end case ;  -- 39 case

case rominput is
          when "000101000" => b40 := '1';
          when OTHERS => b40 := '0';
end case ;
case rominput is
          when "000101001" => b41 := '1';
          when OTHERS => b41 := '0';
end case ;
case rominput is
          when "000101010" => b42 := '1';
          when OTHERS => b42 := '0';
end case ;
case rominput is
          when "000101011" => b43 := '1';
          when OTHERS => b43 := '0';
end case ;
case rominput is
          when "000101100" => b44 := '1';
          when OTHERS => b44 := '0';
end case ;
case rominput is
          when "000101101" => b45 := '1';
          when OTHERS => b45 := '0';
end case ;
case rominput is
          when "000101110" => b46 := '1';
          when OTHERS => b46 := '0';
end case ;
case rominput is
          when "000101111" => b47 := '1';
          when OTHERS => b47 := '0';
end case ;
case rominput is
```

```
divide_double ,
fpoint_add ,
fpoint_divide ,
fpoint_mult ,
fpoint_subtract ,
literal_add ,
literal_double ,
sign_ext_double ,
sign_ext ,
and_op ,
or_op ,
xor_op ,
byte_comp ,
comp ,
bit_comp ,
comp_double ,
logical_comp ,
masked_comp ,
literal_comp ,
float_comp ,
 alg_ldouble_shft ,
alg_left_shft ,
alg_right_shft ,
alg_rdouble_shft ,
cir_left_shft ,
cir_ldouble_shft ,
literal_left_shft ,
literal_ldouble_shft ,
literal_right_shft ,
literal_rdouble_shft ,
logical_right_shft ,
logical_rdouble_shft ,
  stack_get_top ,
  stack_put_top ,
  queue_get_top ,
  queue_put_top ,
  queue_put_bottom ,
   float_to_single ,
   float_to_double ,
   fixed_to_float ,
   fixed_double_to_float ,
   floating_normalize ,
   algebraic_left_quadruple_shift ,
   algebraic_right_quadruple_shift ,
   executive_return ,
   count_ones ,
   biased_fetch ,
   masked_substitute ,
   remote_execute ,
   set_bit ,
   zero_bit ,
   reverse_register ,
   scale_factor ,
      disable_clock_int ,
      disable_rtc_register ,
      disable_monitor_clock ,
      enable_clock_int ,
      enable_clock_and_int ,
      load_rtc_lower ,
      load_double_and_clock ,
      load_enable_monitor_clock ,
      load_clock ,
      reset_bit_timer ,
      diagnostic_jump ,
```



```
      jump_stop_key ,
      jump_stop_key2 ,
      jump_after_stop ,
      load_addr_register ,
      load_physical_addr ,
      load_physical_location ,
      store_physical_location ,
      load_sr1 ,
      load_sr2 ,
       set_bit_indicator ,
       init_processor_int ,
       ipl_failed ,
       load_p ,
       store_clock ,
       store_clock_double ,
       store_sr1 ,
       store_sr2 ,
       store_monitor_clock ,
       fixed_point_double_arccosine ,
       fixed_point_double_arcsine ,
       fixed_point_double_arctan ,
       fixed_point_double_exponent ,
       fixed_point_cross_product ,
       fixed_point_matrix_multiply ,
       fixed_point_double_natural_log ,
       fixed_point_sum_square ,
       fixed_point_double_sine_cosine ,
       fixed_point_double_square_root ,
       decrease_ra_1 ,
       decrease_ra_2 ,
       increase_ra_1 ,
       make_negative ,
       one_complement ,
       make_positive ,
       round_off ,
       two_complement_double ,
       two_complement ,
      jump ,
      jump_bootstrap ,
      jump_on_carry ,
      jump_equal ,
      jump_geater_equal ,
      jump_less ,
      jump_not_equal ,
      jump_over_flow ,
      jump_power_out ,
      jump_link_memory ,
      jump_link_register ,
      jump_negative ,
      jump_positive ,
      jump_not_zero ,
      jump_zero ,
      zero ,
      nzero ,
      pos ,
      neg ,
      carry ,
      ovr ) ;

   variable op : bit_vector(5 downto 0);


BEGIN
```

```
-- $$$$$$$$$$$$$$$$$  Begin decoding the op code format $$$$$$$$$$$$$$$$$$$$$$$$$

 if opdeco_en = '1' then
       op := to_bitvector(inputdata(15 downto 10));

   -- decode instruction set
      case op is
            -- LOAD/STORE
                when op_byte_load(7 downto 2) => byte_load <= '1' ;

                when OTHERS => byte_load <= '0'    ;

                end case ;
      case op is
                when op_byte_load_index(7 downto 2) => byte_load_index <= '1' ;

                when OTHERS =>   byte_load_index <= '0'  ;
                end case ;
      case op is
                when op_byte_store_index(7 downto 2) => byte_store_index <= '1' ;

                when OTHERS =>    byte_store_index <= '0' ;
                end case ;
      case op is
                when op_double_load_index(7 downto 2) => double_load_index <= '1' ;
                when OTHERS =>  double_load_index <= '0'   ;
                end case ;
      case op is
                when op_literal_load(7 downto 2) => literal_load <= '1';
                when OTHERS =>  literal_load <= '0'   ;
                end case ;
      case op is
                when op_load_multiple(7 downto 2) => load_multiple <= '1';
                when OTHERS => load_multiple <= '0'    ;
                end case ;
      case op is
                when op_load_mult_reg(7 downto 2)  => load_mult_reg <= '1';
                when OTHERS =>   load_mult_reg <= '0'  ;
                end case ;
      case op is
                when op_store_mult_reg(7 downto 2)  => store_mult_reg <= '1';
                when OTHERS =>  store_mult_reg <= '0'   ;
                end case ;
      case op is
                when op_word_load_index(7 downto 2)  => word_load_index <= '1';
                when OTHERS =>   word_load_index <= '0'   ;
                end case ;
      case op is
                when op_move_block(7 downto 2) => move_block <= '1';
                when OTHERS => move_block <= '0'    ;
                end case ;
      case op is
                when op_load(7 downto 2) => load <= '1' ;
                when OTHERS =>  load <= '0'    ;
                end case ;
      case op is
                when op_load_bit(7 downto 2) => load_bit <= '1';
                when OTHERS =>  load_bit <= '0'   ;
                end case ;
      case op is
                when op_load_double(7 downto 2) => load_double <= '1';
                when OTHERS => load_double <= '0'    ;
                end case ;
```



```
case op is
          when op_load_mult(7 downto 2) => load_mult <= '1';
          when OTHERS => load_mult <= '0'    ;
          end case ;
case op is
          when op_byte_store(7 downto 2) => byte_store <= '1' ;
          when OTHERS =>  byte_store <= '0'    ;
          end case ;
case op is
          when op_store(7 downto 2) => store <= '1' ;
          when OTHERS =>   store <= '0'    ;
          end case ;
case op is
          when op_store_double(7 downto 2) => store_double <= '1';
          when OTHERS => store_double <= '0'    ;
          end case ;
case op is
          when op_store_address(7 downto 2) => store_address <= '1';
          when OTHERS =>   store_address <= '0'   ;
          end case ;
case op is
          when op_store_zero(7 downto 2) => store_zero <= '1';
          when OTHERS =>  store_zero <= '0'   ;
          end case ;
case op is
          when op_decindex_store(7 downto 2) => decindex_store <= '1';
          when OTHERS =>  decindex_store <= '0'    ;
          end case ;
case op is
          when op_decindex_store_double(7 downto 2) => decindex_store_double <= '1';
          when OTHERS => decindex_store_double <= '0'    ;
          end case ;

case op is
          when op_jump_io(7 downto 2) => jump_io <= '1';
          when OTHERS => jump_io <= '0'    ;
          end case ;
case op is
          when op_jump_link_memory(7 downto 2) => jump_link_memory <= '1';
          when OTHERS =>   jump_link_memory <= '0'  ;
          end case ;
case op is
          when op_jump_link_register(7 downto 2) => jump_link_register <= '1';
          when OTHERS => jump_link_register <= '0'    ;
          end case ;
case op is
          when op_jump_negative(7 downto 2) => jump_negative <= '1';
          when OTHERS => jump_negative <= '0'    ;
          end case ;
case op is
          when op_jump_not_zero(7 downto 2) => jump_not_zero <= '1';
          when OTHERS =>jump_not_zero <= '0'     ;
          end case ;
case op is
          when op_jump_positive(7 downto 2) => jump_positive <= '1';
          when OTHERS =>  jump_positive <= '0'   ;
          end case ;
case op is
          when op_jump_zero(7 downto 2) => jump_zero <= '1';
          when OTHERS => jump_zero <= '0'    ;
          end case ;

-- ARITHMETIC
case op is
```

```
            when op_add(7 downto 2) => add <= '1';
            when OTHERS =>   add <= '0'   ;
            end case ;
case op is
            when op_add_byte(7 downto 2) => add_byte <= '1';
            when OTHERS => add_byte <= '0'    ;
            end case ;
case op is
            when op_add_double(7 downto 2) => add_double <= '1';
            when OTHERS =>  add_double <= '0'    ;
            end case ;
case op is
            when op_subtract(7 downto 2) => subtract <= '1';
            when OTHERS =>   subtract <= '0'  ;
            end case ;
case op is
            when op_subtract_byte(7 downto 2) => subtract_byte <= '1';
            when OTHERS => subtract_byte <= '0'    ;
            end case ;
case op is
            when op_subtract_double(7 downto 2) => subtract_double <= '1';
            when OTHERS =>  subtract_double <= '0'   ;
            end case ;
case op is
            when op_multiply(7 downto 2) => multiply <= '1';
            when OTHERS => multiply <= '0'     ;
            end case ;
case op is
            when op_multiply_double(7 downto 2) => multiply_double <= '1';
            when OTHERS => multiply_double <= '0'    ;
            end case ;
case op is
            when op_divide(7 downto 2) => divide <= '1';
            when OTHERS => divide <= '0';
            end case ;
case op is
            when op_divide_double(7 downto 2) => divide_double <= '1';
            when OTHERS =>  divide_double <= '0'   ;
            end case ;
case op is
            when op_fpoint_add(7 downto 2) => fpoint_add <= '1';
            when OTHERS =>  fpoint_add <= '0'    ;
            end case ;
case op is
            when op_fpoint_divide(7 downto 2) => fpoint_divide <= '1';
            when OTHERS =>  fpoint_divide <= '0'    ;
            end case ;
case op is
            when op_fpoint_mult(7 downto 2) => fpoint_mult <= '1';
            when OTHERS => fpoint_mult <= '0'     ;
            end case ;
case op is
            when op_fpoint_subtract(7 downto 2) => fpoint_subtract <= '1';
            when OTHERS => fpoint_subtract <= '0'    ;
            end case ;
case op is
            when op_literal_add(7 downto 2) => literal_add <= '1';
            when OTHERS => literal_add <= '0'    ;
            end case ;
case op is
            when op_literal_double(7 downto 2) => literal_double <= '1';
            when OTHERS => literal_double <= '0'    ;
            end case ;
case op is
```



```
            when op_sign_ext_double(7 downto 2) => sign_ext_double <= '1';
            when OTHERS =>  sign_ext_double <= '0'   ;
            end case ;
case op is
            when op_sign_ext(7 downto 2) => sign_ext <= '1';
            when OTHERS => sign_ext <= '0'    ;
            end case ;

 -- LOGICAL
case op is
            when op_and(7 downto 2) => and_op <= '1';
            when OTHERS => and_op <= '0'    ;
            end case ;
case op is
            when op_or(7 downto 2) => or_op <= '1';
            when OTHERS => or_op <= '0'    ;
            end case ;
case op is
            when op_xor(7 downto 2) => xor_op <= '1';
            when OTHERS => xor_op <= '0'    ;
            end case ;

 -- COMPARE
case op is
            when op_byte_comp(7 downto 2) => byte_comp <= '1';
            when OTHERS =>  byte_comp <= '0'    ;
            end case ;
case op is
            when op_comp(7 downto 2) => comp <= '1';
            when OTHERS => comp <= '0'    ;
            end case ;
case op is
            when op_bit_comp(7 downto 2) => bit_comp <= '1';
            when OTHERS => bit_comp <= '0'    ;
            end case ;
case op is
            when op_comp_double(7 downto 2) => comp_double <= '1';
            when OTHERS => comp_double <= '0'    ;
            end case ;
case op is
            when op_logical_comp(7 downto 2) => logical_comp <= '1';
            when OTHERS => logical_comp <= '0'    ;
            end case ;
case op is
            when op_masked_comp(7 downto 2) => masked_comp <= '1';
            when OTHERS => masked_comp <= '0'    ;
            end case ;
case op is
            when op_literal_comp(7 downto 2) => literal_comp <= '1';
            when OTHERS => literal_comp <= '0'    ;
            end case ;

 -- SHIFT
case op is
            when op_alg_ldouble_shft(7 downto 2) => alg_ldouble_shft <= '1';
            when OTHERS =>  alg_ldouble_shft <= '0'    ;
            end case ;
case op is
            when op_alg_left_shft(7 downto 2) => alg_left_shft <= '1';
            when OTHERS => alg_left_shft <= '0'    ;
            end case ;
case op is
            when op_alg_right_shft(7 downto 2) => alg_right_shft <= '1';
            when OTHERS => alg_right_shft <= '0'    ;
```

```
            when "010001101" => b141 := '1';
            when OTHERS => b141 := '0';
end case ;
case rominput is
            when "010001110" => b142 := '1';
            when OTHERS => b142 := '0';
end case ;
case rominput is
            when "010001111" => b143 := '1';
            when OTHERS => b143 := '0';
end case ;
case rominput is
            when "010010000" => b144 := '1';
            when OTHERS => b144 := '0';
end case ;
case rominput is
            when "010010001" => b145 := '1';
            when OTHERS => b145 := '0';
end case ;
case rominput is
            when "010010010" => b146 := '1';
            when OTHERS => b146 := '0';
end case ;
case rominput is
            when "010010011" => b147 := '1';
            when OTHERS => b147 := '0';
end case ;
case rominput is
            when "010010100" => b148 := '1';
            when OTHERS => b148 := '0';
end case ;
case rominput is
            when "010010101" => b149 := '1';
            when OTHERS => b149 := '0';
end case ;  -- 149 cases

case rominput is
            when "010010110" => b150 := '1';
            when OTHERS => b150 := '0';
end case ;
case rominput is
            when "010010111" => b151 := '1';
            when OTHERS => b151 := '0';
end case ;
case rominput is
            when "010011000" => b152 := '1';
            when OTHERS => b152 := '0';
end case ;
case rominput is
            when "010011001" => b153 := '1';
            when OTHERS => b153 := '0';
end case ;
case rominput is
            when "010011010" => b154 := '1';
            when OTHERS => b154 := '0';
end case ;
case rominput is
            when "010011011" => b155 := '1';
            when OTHERS => b155 := '0';
end case ;
case rominput is
            when "010011100" => b156 := '1';
            when OTHERS => b156 := '0';
end case ;
```



```
case rominput is
            when "010011101" => b157 := '1';
            when OTHERS => b157 := '0';
end case ;
case rominput is
            when "010011110" => b158 := '1';
            when OTHERS => b158 := '0';
end case ;
case rominput is
            when "010011111" => b159 := '1';
            when OTHERS => b159 := '0';
end case ;  -- 159 cases

case rominput is
            when "010100000" => b160 := '1';
            when OTHERS => b160 := '0';
end case ;
case rominput is
            when "010100001" => b161 := '1';
            when OTHERS => b161 := '0';
end case ;
case rominput is
            when "010100010" => b162 := '1';
            when OTHERS => b162 := '0';
end case ;
case rominput is
            when "010100011" => b163 := '1';
            when OTHERS => b163 := '0';
end case ;
case rominput is
            when "010100100" => b164 := '1';
            when OTHERS => b164 := '0';
end case ;
case rominput is
            when "010100101" => b165 := '1';
            when OTHERS => b165 := '0';
end case ;
case rominput is
            when "010100110" => b166 := '1';
            when OTHERS => b166 := '0';
end case ;
case rominput is
            when "010100111" => b167 := '1';
            when OTHERS => b167 := '0';
end case ;
case rominput is
            when "010101000" => b168 := '1';
            when OTHERS => b168 := '0';
end case ;
case rominput is
            when "010101001" => b169 := '1';
            when OTHERS => b169 := '0';
end case ;  -- 169 cases

case rominput is
            when "010101010" => b170 := '1';
            when OTHERS => b170 := '0';
end case ;
case rominput is
            when "010101011" => b171 := '1';
            when OTHERS => b171 := '0';
end case ;
case rominput is
            when "010101100" => b172 := '1';
```

```
            when OTHERS => b172 := '0';
end case ;
case rominput is
            when "010101101" => b173 := '1';
            when OTHERS => b173 := '0';
end case ;
case rominput is
            when "010101110" => b174 := '1';
            when OTHERS => b174 := '0';
end case ;
case rominput is
            when "010101111" => b175 := '1';
            when OTHERS => b175 := '0';
end case ;
case rominput is
            when "010110000" => b176 := '1';
            when OTHERS => b176 := '0';
end case ;
case rominput is
            when "010110001" => b177 := '1';
            when OTHERS => b177 := '0';
end case ;
case rominput is
            when "010110010" => b178 := '1';
            when OTHERS => b178 := '0';
end case ;
case rominput is
            when "010110011" => b179 := '1';
            when OTHERS => b179 := '0';
end case ;  -- 179 cases

case rominput is
            when "010110100" => b180 := '1';
            when OTHERS => b180 := '0';
end case ;
case rominput is
            when "010110101" => b181 := '1';
            when OTHERS => b181 := '0';
end case ;
case rominput is
            when "010110110" => b182 := '1';
            when OTHERS => b182 := '0';
end case ;
case rominput is
            when "010110111" => b183 := '1';
            when OTHERS => b183 := '0';
end case ;
case rominput is
            when "010111000" => b184 := '1';
            when OTHERS => b184 := '0';
end case ;
case rominput is
            when "010111001" => b185 := '1';
            when OTHERS => b185 := '0';
end case ;
case rominput is
            when "010111010" => b186 := '1';
            when OTHERS => b186 := '0';
end case ;
case rominput is
            when "010111011" => b187 := '1';
            when OTHERS => b187 := '0';
end case ;
case rominput is
```



```
            when "010111100" => b188 := '1';
            when OTHERS => b188 := '0';
end case ;
case rominput is
            when "010111101" => b189 := '1';
            when OTHERS => b189 := '0';
end case ;  -- 189 cases

case rominput is
            when "010111110" => b190 := '1';
            when OTHERS => b190 := '0';
end case ;
case rominput is
            when "010111111" => b191 := '1';
            when OTHERS => b191 := '0';
end case ;
case rominput is
            when "011000000" => b192 := '1';
            when OTHERS => b192 := '0';
end case ;
case rominput is
            when "011000001" => b193 := '1';
            when OTHERS => b193 := '0';
end case ;
case rominput is
            when "011000010" => b194 := '1';
            when OTHERS => b194 := '0';
end case ;
case rominput is
            when "011000011" => b195 := '1';
            when OTHERS => b195 := '0';
end case ;
case rominput is
            when "011000100" => b196 := '1';
            when OTHERS => b196 := '0';
end case ;
case rominput is
            when "011000101" => b197 := '1';
            when OTHERS => b197 := '0';
end case ;
case rominput is
            when "011000110" => b198 := '1';
            when OTHERS => b198 := '0';
end case ;
case rominput is
            when "011000111" => b199 := '1';
            when OTHERS => b199 := '0';
end case ;  -- 199 cases


--- done 200 combination
case rominput is
            when "011001000" => b200 := '1';
            when OTHERS => b200 := '0';
end case ;
case rominput is
            when "011001001" => b201 := '1';
            when OTHERS => b201 := '0';
end case ;
case rominput is
            when "011001010" => b202 := '1';
            when OTHERS => b202 := '0';
end case ;
case rominput is
```

```
            when "011001011" => b203 := '1';
            when OTHERS => b203 := '0';
end case ;
case rominput is
            when "011001100" => b204 := '1';
            when OTHERS => b204 := '0';
end case ;
case rominput is
            when "011001101" => b205 := '1';
            when OTHERS => b205 := '0';
end case ;
case rominput is
            when "011001110" => b206 := '1';
            when OTHERS => b206 := '0';
end case ;
case rominput is
            when "011001111" => b207 := '1';
            when OTHERS => b207 := '0';
end case ;
case rominput is
            when "011010000" => b208 := '1';
            when OTHERS => b208 := '0';
end case ;
case rominput is
            when "011010001" => b209 := '1';
            when OTHERS => b209 := '0';
end case ; -- 209 cases

case rominput is
            when "011010010" => b210 := '1';
            when OTHERS => b210 := '0';
end case ;
case rominput is
            when "011010011" => b211 := '1';
            when OTHERS => b211 := '0';
end case ;
case rominput is
            when "011010100" => b212 := '1';
            when OTHERS => b212 := '0';
end case ;
case rominput is
            when "011010101" => b213 := '1';
            when OTHERS => b213 := '0';
end case ;
case rominput is
            when "011010110" => b214 := '1';
            when OTHERS => b214 := '0';
end case ;
case rominput is
            when "011010111" => b215 := '1';
            when OTHERS => b215 := '0';
end case ;
case rominput is
            when "011011000" => b216 := '1';
            when OTHERS => b216 := '0';
end case ;
case rominput is
            when "011011001" => b217 := '1';
            when OTHERS => b217 := '0';
end case ;
case rominput is
            when "011011010" => b218 := '1';
            when OTHERS => b218 := '0';
end case ;
```



```
case rominput is
            when "011011011" => b219 := '1';
            when OTHERS => b219 := '0';
end case ; -- 219 cases

case rominput is
            when "011011100" => b220 := '1';
            when OTHERS => b220 := '0';
end case ;
case rominput is
            when "011011101" => b221 := '1';
            when OTHERS => b221 := '0';
end case ;
case rominput is
            when "011011110" => b222 := '1';
            when OTHERS => b222 := '0';
end case ;
case rominput is
            when "011011111" => b223 := '1';
            when OTHERS => b223 := '0';
end case ;
case rominput is
            when "011100000" => b224 := '1';
            when OTHERS => b224 := '0';
end case ;
case rominput is
            when "011100001" => b225 := '1';
            when OTHERS => b225 := '0';
end case ;
case rominput is
            when "011100010" => b226 := '1';
            when OTHERS => b226 := '0';
end case ;
case rominput is
            when "011100011" => b227 := '1';
            when OTHERS => b227 := '0';
end case ;
case rominput is
            when "011100100" => b228 := '1';
            when OTHERS => b228 := '0';
end case ;
case rominput is
            when "011100101" => b229 := '1';
            when OTHERS => b229 := '0';
end case ;  -- 229 cases

case rominput is
            when "011100110" => b230 := '1';
            when OTHERS => b230 := '0';
end case ;
case rominput is
            when "011100111" => b231 := '1';
            when OTHERS => b231 := '0';
end case ;
case rominput is
            when "011101000" => b232 := '1';
            when OTHERS => b232 := '0';
end case ;
case rominput is
            when "011101001" => b233 := '1';
            when OTHERS => b233 := '0';
end case ;
case rominput is
            when "011101010" => b234 := '1';
```

```
            when OTHERS => b234 := '0';
end case ;
case rominput is
            when "011101011" => b235 := '1';
            when OTHERS => b235 := '0';
end case ;
case rominput is
            when "011101100" => b236 := '1';
            when OTHERS => b236 := '0';
end case ;
case rominput is
            when "011101101" => b237 := '1';
            when OTHERS => b237 := '0';
end case ;
case rominput is
            when "011101110" => b238 := '1';
            when OTHERS => b238 := '0';
end case ;
case rominput is
            when "011101111" => b239 := '1';
            when OTHERS => b239 := '0';
end case ;  -- 239 cases

case rominput is
            when "011110000" => b240 := '1';
            when OTHERS => b240 := '0';
end case ;
case rominput is
            when "011110001" => b241 := '1';
            when OTHERS => b241 := '0';
end case ;
case rominput is
            when "011110010" => b242 := '1';
            when OTHERS => b242 := '0';
end case ;
case rominput is
            when "011110011" => b243 := '1';
            when OTHERS => b243 := '0';
end case ;
case rominput is
            when "011110100" => b244 := '1';
            when OTHERS => b244 := '0';
end case ;
case rominput is
            when "011110101" => b245 := '1';
            when OTHERS => b245 := '0';
end case ;
case rominput is
            when "011110110" => b246 := '1';
            when OTHERS => b246 := '0';
end case ;
case rominput is
            when "011110111" => b247 := '1';
            when OTHERS => b247 := '0';
end case ;
case rominput is
            when "011111000" => b248 := '1';
            when OTHERS => b248 := '0';
end case ;
case rominput is
            when "011111001" => b249 := '1';
            when OTHERS => b249 := '0';
end case ; -- 249 cases
```



```
case rominput is
            when "011111010" => b250 := '1';
            when OTHERS => b250 := '0';
end case ;
case rominput is
            when "011111011" => b251 := '1';
            when OTHERS => b251 := '0';
end case ;
case rominput is
            when "011111100" => b252 := '1';
            when OTHERS => b252 := '0';
end case ;
case rominput is
            when "011111101" => b253 := '1';
            when OTHERS => b253 := '0';
end case ;
case rominput is
            when "011111110" => b254 := '1';
            when OTHERS => b254 := '0';
end case ;
case rominput is
            when "011111111" => b255 := '1';
            when OTHERS => b255 := '0';
end case ;
case rominput is
            when "100000000" => b256 := '1';
            when OTHERS => b256 := '0';
end case ;
case rominput is
            when "100000001" => b257 := '1';
            when OTHERS => b257 := '0';
end case ;
case rominput is
            when "100000010" => b258 := '1';
            when OTHERS => b258 := '0';
end case ;
case rominput is
            when "100000011" => b259 := '1';
            when OTHERS => b259 := '0';
end case ; -- 259 cases

case rominput is
            when "100000100" => b260 := '1';
            when OTHERS => b260 := '0';
end case ;
case rominput is
            when "100000101" => b261 := '1';
            when OTHERS => b261 := '0';
end case ;
case rominput is
            when "100000110" => b262 := '1';
            when OTHERS => b262 := '0';
end case ;
case rominput is
            when "100000111" => b263 := '1';
            when OTHERS => b263 := '0';
end case ;
case rominput is
            when "100001000" => b264 := '1';
            when OTHERS => b264 := '0';
end case ;
case rominput is
            when "100001001" => b265 := '1';
            when OTHERS => b265 := '0';
```

```
end case ;
case rominput is
          when "100001010" => b266 := '1';
          when OTHERS => b266 := '0';
end case ;
case rominput is
          when "100001011" => b267 := '1';
          when OTHERS => b267 := '0';
end case ;
case rominput is
          when "100001100" => b268 := '1';
          when OTHERS => b268 := '0';
end case ;
case rominput is
          when "100001101" => b269 := '1';
          when OTHERS => b269 := '0';
end case ;  -- 269 cases

case rominput is
          when "100001110" => b270 := '1';
          when OTHERS => b270 := '0';
end case ;
case rominput is
          when "100001111" => b271 := '1';
          when OTHERS => b271 := '0';
end case ;
case rominput is
          when "100010000" => b272 := '1';
          when OTHERS => b272 := '0';
end case ;
case rominput is
          when "100010001" => b273 := '1';
          when OTHERS => b273 := '0';
end case ;
case rominput is
          when "100010010" => b274 := '1';
          when OTHERS => b274 := '0';
end case ;
case rominput is
          when "100010011" => b275 := '1';
          when OTHERS => b275 := '0';
end case ;
case rominput is
          when "100010100" => b276 := '1';
          when OTHERS => b276 := '0';
end case ;
case rominput is
          when "100010101" => b277 := '1';
          when OTHERS => b277 := '0';
end case ;
case rominput is
          when "100010110" => b278 := '1';
          when OTHERS => b278 := '0';
end case ;
case rominput is
          when "100010111" => b279 := '1';
          when OTHERS => b279 := '0';
end case ;  -- 279 cases

case rominput is
          when "100011000" => b280 := '1';
          when OTHERS => b280 := '0';
end case ;
case rominput is
```



```
          when "100011001" => b281 := '1';
          when OTHERS => b281 := '0';
end case ;
case rominput is
          when "100011010" => b282 := '1';
          when OTHERS => b282 := '0';
end case ;
case rominput is
          when "100011011" => b283 := '1';
          when OTHERS => b283 := '0';
end case ;
case rominput is
          when "100011100" => b284 := '1';
          when OTHERS => b284 := '0';
end case ;
case rominput is
          when "100011101" => b285 := '1';
          when OTHERS => b285 := '0';
end case ;
case rominput is
          when "100011110" => b286 := '1';
          when OTHERS => b286 := '0';
end case ;
case rominput is
          when "100011111" => b287 := '1';
          when OTHERS => b287 := '0';
end case ;
case rominput is
          when "100100000" => b288 := '1';
          when OTHERS => b288 := '0';
end case ;
case rominput is
          when "100100001" => b289 := '1';
          when OTHERS => b289 := '0';
end case ; -- 289 cases

case rominput is
          when "100100010" => b290 := '1';
          when OTHERS => b290 := '0';
end case ;
case rominput is
          when "100100011" => b291 := '1';
          when OTHERS => b291 := '0';
end case ;
case rominput is
          when "100100100" => b292 := '1';
          when OTHERS => b292 := '0';
end case ;
case rominput is
          when "100100101" => b293 := '1';
          when OTHERS => b293 := '0';
end case ;
case rominput is
          when "100100110" => b294 := '1';
          when OTHERS => b294 := '0';
end case ;
case rominput is
          when "100100111" => b295 := '1';
          when OTHERS => b295 := '0';
end case ;
case rominput is
          when "100101000" => b296 := '1';
          when OTHERS => b296 := '0';
end case ;
```

```
        case rominput is
                  when "100101001" => b297 := '1';
                  when OTHERS => b297 := '0';
        end case ;
        case rominput is
                  when "100101010" => b298 := '1';
                  when OTHERS => b298 := '0';
        end case ;
        case rominput is
                  when "100101011" => b299 := '1';
                  when OTHERS => b299 := '0';
        end case ; -- 299 cases

 --- done 300 combination


   rom_addr <= b299&b298&b297&b296&b295&b294&b293&b292&b291&b290&b289&b288&b287&b286&b285&b284&



 -- ***********  Done decode for instruction code *************
 -- $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

--    end if ; -- opdeco_en = '1'

END PROCESS;
END behaviour ;
```

Appendix B - PowerPC instruction sequence

AN/AYK-14 Instruction Set						PowerPC Instruction set									
Mnemonic	Format/Opcode					Mnemonic	0-5	6-10	11-15	16-20	21	22-30	31		
	8	4	4	#				D	A	B	OE		Rc		
DRORa	O8	a	9		7	LHA Ra,d(r0)	42	A		0		d			
					6	ADDMEO. Ra, Ra	31	A	A		0	1	234	1	
					5	RLWINM Ra, Ra,SH=16,MB=0,ME=15	21	A	A		16	0	15	0	
					4	STW Rs, d(r0)	36	A		0		d			
					3	MCRXR CRFD	31		1	0	0		512	0	H
					2	MFCD R31	31	D		0	0		19	0	
					1	STW Rs, d(r0)	36		31	0		31			H
IRTRa	O8	a	A		8	LWZ Ra,d(r0)	32	A		0		d			
					7	LWZ R31,d(r0)	32		31	0		2			
					6	RLWINM R31, R31,SH=16,MB=0,ME=15	21		31	31	16	0	15	0	
					5	ADDIC. Ra, Ra	31	A	A			2			
					4	STW RA, d(r0)	36	A		0		d			H
					3	MCRXR CRFD	31		1	0	0		512	0	
					2	MFCD R31	31	D		0	0		19	0	
					1	STW R31, d(r0)	36		31	0		31			H
DRTRa	O8	a	B		8	LHA Ra,d(r0)	42	A		0		d			
					7	ADDMEO. Ra, Ra	31	A	A		0	1	234	1	
					6	ADDMEO. Ra, Ra	31	A	A		0	1	234	1	
					5	RLWINM Ra, Ra,SH=16,MB=0,ME=15	21	A	A		16	0	15	0	
					4	STW Ra, d(r0)	36	A		0		d			H
					3	MCRXR CRFD	31		1	0	0		512	0	
					2	MFCD R31	31	D		0	0		19	0	
					1	STW Rs, d(r0)	36		31	0		31			H
SFR	10	a	3		10	LWZ Ra,d(r0)	32	A		0		d			
					9	ADDIC R31, Ra, d(=0)	12		31	A			SIMM=1		
					8	BC Bo,Bi, targ_addr	16	H'0D		H'00		BD=10			
					7	ANDC R31, R31, R31	31		31	31	31	0	60	0	
					6	NOR R31, R31, Ra	31		31	A	31		124	0	
					5	CNTLZW Ra+2,R31	31		31	A+2	H'00		26	0	
					4	ADDMEO. Ra+2, Ra+2	31	A+2		A+2	0	1	234	1	
					3	SLW Ra,Ra,Ra+2	31	A		A	A+2		24	1	
					2	STW Ra,d(r0)	36	A		0		d			D
					1	STW Ra+2, d(r0)	36	A+2		0		d			L
SBR a, m	14	a	m		5	LHA Ra,d(r0)	42	A		0		d			
					4	ORI Ra, Ra,UIMM=m	24	A		A		UIMM=m			
					3	MCRXR crfD	31	crfD00		H'00	H'00	512		0	
					2	SRAWI. Ra,Ra,SH	31	A		A	H'00	824		1	
					1	STW Ra, d(r0)	36	A		0		d			L
ZBR a, m	18	a	m		4	LHA Ra,d(r0)	42	A		0		d			
	18	a	m		3	ANDI Ra,Rs,UIMM=~m	28	A		A		UIMM=~m			
	18	a	m		2	RLWINM Ra, Ra,SH=16,MB=0,ME=15	20	A		A	16	0	15	1	
	18	a	m		1	STW Ra, d(r0)	36	A		0		d			H
LRSR a, m	20	a	m		6	LWA Ra,d(r0)	42	A		0		d			
LRS a, y, m	22	a	m		5	LHA Rm,d(r0)	32	M		0		d			
LRDR a, m	28	a	m		4	SRW. Ra, Ra, R31	31	A		A	31		536	1	
LRD a, y, m	2A	a	m		3	STW Ra, d(r0)	36	A		0		d			L,D
					2	MFCD R31	31	D		0	0		19	0	
					1	STW R31, d(r0)	36		31	0		31			H
ARSR a, m	24	a	m		6	LWA Ra,d(r0)	42	A		0		d			
ARS a, y, m	26	a	m		5	LHA Rm,d(r0)	32	M		0		d			
ARDR a, m	2C	a	m		4	SRAW. Ra, Ra, R31	31	A		A	31		792	1	
ARD a, y, m	2E	a	m		3	STW Ra, d(r0)	36	A		0		d			L,D
					2	MFCD R31	31	D		0	0		19	0	
					1	STW R31, d(r0)	36		31	0		31			H
ALSR	30	a	m		6	LWA Ra,d(r0)	42	A		0		d			
ALS a, y, m	32	a	m		5	LHA Rm,d(r0)	32	M		0		d			
ALDR a, m	38	a	m		4	SLW. Ra, Ra, R31	31	A		A	31		24	1	
ALD a, y, m	3A	a	m		3	STW Ra, d(r0)	36	A		0		d			L,D
					2	MFCD R31	31	D		0	0		19	0	
					1	STW R31, d(r0)	36		31	0		31			H
CLSR a, m	34	a	m		8	LHA Ra,d(r0)	32	A		0		d			
CLS a, y, m	36	a	m		7	RLWINM R31, Ra,SH=16,MB=0,ME=15	21		31	A	16	0	15	0	
					6	OR Ra, Ra, R31	31	A		A	31		444	0	
					5	LHA Rm,d(r0)	32	M		0		d			
					4	RLWNM Ra, Ra,Rm,MB=16,ME=31	23	A		A	M	16	32	0	
					3	STW Ra, d(r0)	36	A		0		d			L
					2	MFCD R31	31	D		0	0		19	0	
					1	STW R31, d(r0)	36		31	0		31			H
CLDR a, m	3C	a	m		6	LWA Ra,d(r0)	42	A		0		d			
CLD a, y, m	3E	a	m		5	LHA Rm,d(r0)	32	M		0		d			
					4	RLWNM. Ra, Ra, R31,16,32	23	A		A	31	16	32	1	
					3	STW Ra, d(r0)	36	A		0		d			D
					2	MFCD R31	31	D		0	0		19	0	
					1	STW R31, d(r0)	36		31	0		31			H
SUR a, m	40	a	m		7	LWZ Ra,d(r0)	32	A		0		d			

SUI a, m	41	a	m	6	LWZ Rm,d(r0)	32	m		0		d+1														
SUK a, m	42	a	m	5	SUBFO. Ra,Rm,Ra	31	A		M		A		1		40		1								
SU a, m	43	a	m	4	STW Ra, d(r0)	36	A		0		d														H
SUDR a, m	44	a	m	3	MCRXR CRFD	31		1		0		0			512		0		H						
SUDI a, m	45	a	m	2	MFCD R31	31	D			0		0			19		0								
SUD a, m	47	a	m	1	STW Rs, d(r0)	36		31		0			31						H						
AR a, m	48	a	m	7	LWZ Ra,d(r0)	32	A			0			d												
AI a, m	49	a	m	6	LWZ Rm,d(r0)	32	m			0			d+1												
AK a, y, m	4A	a	m	5	ADDCO. Ra, Ra, Rm	31	A		A		M		1		10		1								
A a, y, m	4B	a	m	4	STW Ra, d(r0)	36	A			0			d												H
ARD a, m	4C	a	m	3	MCRXR CRFD	31		1		0		0			512		0		H						
ADI a, m	4D	a	m	2	MFCD R31	31	D			0		0			19		0								
AD a, m	4F	a	m	1	STW Rs, d(r0)	36		31		0			31						H						
CR a, m	50	a	m	5	LHA Rm,d(r0)	42	m			0			d												
CI a, m	51	a	m	4	LHA Ra,d(r0)	42	A			0			d+1												
CK a, y, m	52	a	m	3	CMP CRFD,0,Ra,Rm	31	CR0 0(		A		M				0										
C a, y, m	53	a	m	2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
CDR a, m	54	a	m	5	LWZ Rm,d(r0)	42	m			0			d												
CDI a, m	55	a	m	4	LWZ Ra,d(r0)	42	A			0			d+1												
CD a, y, m	57	a	m	3	CMP CRFD,0,Ra,Rm	31	CR0 0(		A		M				0										
				2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
MR a, m	58	a	m	6	LWZ Ra+1,d(r0)	32	A+1			0			d												
MI a, m	59	a	m	5	LWZ Rm,d(r0)	32	M			0			d+1												
MK a, y, m	5A	a	m	4	MULHW. Ra,Ra+1,Rm	31	A		A+1		M		0		75		1								
MI a, m	5B	a	m	3	STW Ra, d(r0)	36	A			0			d												D
				2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
DR a, m	5C	a	m	7	LWZ Ra,a+1,d(r0)	32	A			0			d												
DI a, m	5D	a	m	6	LHA Rm,d(r0)	42	A			0			d+1												
DK a, y, m	5E	a	m	5	DIVO. Ra+1,R31,Rm	31	A+1			31		M		1	331		1								
D a, y, m	5F	a	m	4	RLWINM R30, Ra+1,SH=0,MB=16,ME=31	21	A+1			30		0		16	31		0								
				3	MULLU. Ra,R30,Rm	31	A			30		M		1	235		0								
				2	STW Ra, d(r0)	36	A			0			d												H
				1	STW Ra, d(r0)	36	A			0			d												H
ANDR a, m	60	a	m	6	LWZ Ra,d(r0)	32	A			0			d												
ANDI a, m	61	a	m	5	LWZ Rm,d(r0)	32	M			0			d+1												
ANDK a, y, m	62	a	m	4	AND. Ra,Ra,Rm	31	A		A		M		0		28		1								
AND a, y, m	63	a	m	3	STW Ra, d(r0)	36	A			0			d												D
				2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
ORR a, m	64	a	m	6	LWZ Ra,d(r0)	32	A			0			d												
ORI a, m	65	a	m	5	LWZ Rm,d(r0)	32	M			0			d+1												
ORK a, y, m	66	a	m	4	OR. Ra,Ra,Rm	31	A		A		M		0		444		1								
OR a, y, m	67	a	m	3	STW Ra, d(r0)	36	A			0			d												D
				2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
XORR a, m	68	a	m	6	LWZ Ra,d(r0)	32	A			0			d												
XORI a, m	69	a	m	5	LWZ Rm,d(r0)	32	M			0			d+1												
XORK a, y, m	6A	a	m	4	XOR. Ra,Ra,Rm	31	A		A		M		0		316		1								
XOR a, y, m	6B	a	m	3	STW Ra, d(r0)	36	A			0			d												D
				2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
MSR a, m	6C	a	m	11	LHA Ra+1,d(r0)	42	A+1			0			d												
MSI a, m	6D	a	m	10	LHA Ra,d(r0)	42	A			0			d+1												
MSK a, y, m	6E	a	m	9	LHA Rm,d(r0)	42	M			0			d+2												
MS a, y, m	6F	a	m	8	RLWINM R30, Ra+1,SH=0,MB=ME=32-n	21	A+1			30		0	32-n		32-n		1								
				7	BC Bo,Bi, targ_addr	16		12		2			BD=00												
				6	ANDI. R31,Ra,UIMM=-n	28	A			31			UIMM=-n												
				5	ANDI. R30,Rm,UIMM=n	28	M			30			UIMM=n												
				4	OR. Ra, R30,R31	31	H'1E		A		H'1F				444		1								
				3	STW Ra, d(r0)	36	A			0			d												D
				2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
CMR a, m	70	a	m	8	LHA Ra+1,d(r0)	42	A+1			0			d												
CMI a, m	71	a	m	7	LHA Ra,d(r0)	42	A			0			d+1												
CMK a, m	72	a	m	6	LHA Rm,d(r0)	42	M			0			d+2												
CM a, m	73	a	m	5	AND R31,Ra,Ra+1	31	A		H'1F		A+1				28		0								
				4	AND R30,Rm,Ra+1	31	M		H'1F		A+1				28		0								
				3	CMP CRFD,0,R30,R31	31	CR0 0(		H'1E		H'1F				0		0								
				2	MFCD R31	31	D			0		0			19		0								
				1	STW Rs, d(r0)	36		31		0			31						H						
FSUR a, m	A0	a	m	6	LFD frA, d(r0)	50	fra			0			d												
FSUI a, m	A1	a	m	5	LFD frM, d(r0)	50	frm			0			d												
FSU a, y, m	A3	a	m	4	FSUB. frA, frA, frM	63	fra		fra		frm		0		20		1								
				3	MFFS fr31	63	fr31			0		0			583		0								
				2	STTD fra, d(r0)	54	fra			0			d												
				1	STTD fr31, d(r0)	54	fr31			0			31												
FAR a, m	A4	a	m	6	LFD frA, d(r0)	50	fra			0			d												

FAI a, m	A5	a	m		5	LFD frM, d(r0)	50	frm	0		d			
FA a, y, m	A7	a	m		4	FADD. frA, frA, frM	63	fra	fra	frm	0	21	1	
					3	MFFS fr31	63	fr31	0	0		583	0	
					2	STTD fra, d(r0)	54	fra	0		d			
					1	STTD fr31, d(r0)	54	fr31	0		31			
FMR a, m	A8	a	m		6	LFD frA, d(r0)	50	fra	0		d			
FMI a, m	A9	a	m		5	LFD frM, d(r0)	50	frm	0		d			
FM a, y, m	AB	a	m		4	FMUL. frA, frA, frM	63	fra	fra	0	frm	25	1	
					3	MFFS fr31	63	fr31	0	0		583	0	
					2	STTD fra, d(r0)	54	fra	0		d			
					1	STTD fr31, d(r0)	54	fr31	0		31			
FDR a, m	AC	a	m		6	LFD frA, d(r0)	50	fra	0		d			
FDI a, m	AD	a	m		5	LFD frM, d(r0)	50	frm	0		d			
FD a, y, m	AF	a	m		4	FDIV. frA, frA, frM	63	fra	fra	frm		25	1	
					3	MFFS fr31	63	fr31	0	0		583	0	
					2	STTD fra, d(r0)	54	fra	0		d			
					1	STTD fr31, d(r0)	54	fr31	0		31			
MDR a; MD	B8	a	m		8	LWZ Ra,a+1,d(r0)	32	A	0		d			
MDI a, MD	B9	a	m		7	LWZ Rm,m+1,d(r0)	32	M	0		d+1			
MD a, y, MD	BB	a	m		6	MULLW. R29,Ra,Rm	31	29	A	M	0	235	0	
					5	MULHW. Ra,Ra,Rm	31	A	A	M	0	75	1	
					4	STW R29, d(r0)	36	29	0		d			D
					3	STW Ra, d(r0)	36	A	0		d			D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW Rs, d(r0)	36	TMP	0		31			H
DDR a, m	BC	a	m		15	LWZ Ra,a+1,d(r0)	32	A	0		d			
DDI a, m	BD	a	m		14	LWZ Rm,m+1,d(r0)	32	M	0		d+1			
DD a, m	BF	a	m		13	LWZ Ra+2,a+3,d(r0)	32	A +2	0		d+2			
					12	ANDC R31,R31,R31	31	31	31	31		60	0	
					11	ANDC R30,R30,R30	31	30	30	30		60	0	
					10	ORI R31, R31, 32	24	31	31			32		
					9	DIVWO. R29, Ra,Rm	31	29	A	M	1	491	1	
					8	MULL R29, R29, Rm	31	29	29	M	0	235	0	
					7	SUB R28, R29, Ra	31	28	29	A	0	40	0	
					6	CNTLZW R27, R28	31	28	27	0		26	0	
					5	ADD R30, R30, R27	31	30	30	27	0	266	0	
					4	CMP crf0, R30, R31	31	0	30	31		0	0	
					3	BC	16							
					2	SUB R30, R30, R31	31	30	30	31	0	40	0	
					1	SLWR								
LLRS a, m	C0	a	m		7	LWA Ra,d(r0)	42	A	0		d			
LLRD a, m	C2	a	m		6	ANDC R31,R31,R31	31	31	31	31		60	0	
					5	ORI R31, R31, m	24	31	31			m		
					4	SRW. Ra, Ra, R31	31	A	A	31		536	1	
					3	STW Ra, d(r0)	36	A	0		d			L,D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LARS a, m	C1	a	m		7	LWA Ra,d(r0)	42	A	0		d			
LARD a, m	C3	a	m		6	ANDC R31,R31,R31	31	31	31	31		60	0	
					5	ORI R31, R31, m	24	31	31			m		
					4	SRAW. Ra, Ra, R31	31	A	A	31		792	1	
					3	STW Ra, d(r0)	36	A	0		d			L,D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LALS a, m	C4	a	m		7	LWA Ra,d(r0)	42	A	0		d			
LALD a, m	C6	a	m		6	ANDC R31,R31,R31	31	31	31	31		60	0	
					5	ORI R31, R31, m	24	31	31			m		
					4	SLW. Ra, Ra, R31	31	A	A	31		24	1	
					3	STW Ra, d(r0)	36	A	0		d			L,D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LCLS a, m	C5	a	m		9	LHA Ra,d(r0)	32	A	0		d			
					8	RLWINM R31, Ra,SH=16,MB=0,ME=15	21	31	A	16	0	15	0	
					7	OR Ra, Ra, R31	31	A	A	31		444	0	
					6	ANDC R30,R30,R30	31	31	31	31		60	0	
					5	ORI R30, R30, m	24	31	31			m		
					4	RLWNM Ra, Ra,SH=0,MB=16,ME=31	21	A+1	A+1	H00	H10	H1F	0	
					3	STW Ra, d(r0)	36	A	0		d			L
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LCLD a, m	C7	a	m		7	LWA Ra,d(r0)	42	A	0		d			
					6	ANDC R30,R30,R30	31	31	31	31		60	0	
					5	ORI R30, R30, m	24	31	31			m		
					4	RLWNM. Ra, Ra, R31,16,32	23	A	A	31	16	32	1	
					3	STW Ra, d(r0)	36	A	0		d			D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LSU a, m	C8	a	m		6	LHA Ra,d(r0)	32	A	0		d			
					5	SUBFIC Ra, Ra,SIMM=m	8	A	A		SIMM=m			
					4	NEGO. Ra, Ra	31	A	A	H'00	1	104	1	
					3	STW Ra, d(r0)	36	A	0		d			L
					2	MFCD R31	31	D	0	0		19	0	

					1	STW R31, d(r0)	36	31	0		31			H
LSUD	a, m	C9	a	m	6	LWA Ra,d(r0)	42	A	0		d			
					5	SUBFIC Ra, Ra,SIMM=m	8	A	A		SIMM=m			
					4	NEGO. Ra, Ra	31	A	A	H'00	1	104	1	
					3	STW Ra, d(r0)	36	A	0		d			D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LA	a, m	CA	a	m	8	LwA Ra,d(r0)	4	A	0		d			
LAD	a, m	CB	a	m	7	ANDC R30,R30,R30	31	31	31	31		60	0	
					6	ADDIC R30, R30, Y	12	30	30		SIMM=m			
					5	RLWINM R30, R30,SH=16,MB=0,ME=15	21	30	30	16	0	15	0	
					4	ADDO. Ra, Ra, R30	31	A	A	30	1	266	1	
					3	STW Ra, d(r0)	36	A	0		d			H,D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LC	a, m	CD	a	m	4	LHA Ra,d(r0)	42	A	0		d+1			
					3	CMPI CRFD,0,Ra,SIMM=m	31	CR0 0(	A		simm=			
					2	MFCD R31	31	D	0	0		19	0	
					1	STW Rs, d(r0)	36	31	0		31			H
LMUL	a, m	CE	a	m	8	LwA Ra+1,d(r0)	4	A	0		d			
					7	ANDC R30,R30,R30	31	31	31	31		60	0	
					6	ADDIC R30, R30, Y	12	30	30		SIMM=m			
					5	RLWINM R30, R30,SH=16,MB=0,ME=15	21	30	30	16	0	15	0	
					4	MULHW. Ra, Ra, R30	31	A	A	30	0	75	1	
					3	STW Ra, d(r0)	36	A	0		d			D
					2	MFCD R31	31	D	0	0		19	0	
					1	STW R31, d(r0)	36	31	0		31			H
LDIV	a, m	CF	a	m	8	LwA Ra+1,d(r0)	4	A	0		d			
					7	ANDC R30,R30,R30	31	31	31	31		60	0	
					6	ADDIC R30, R30, Y	12	30	30		SIMM=m			
					5	RLWINM R30, R30,SH=16,MB=0,ME=15	21	30	30	16	0	15	0	
					5	DIVO. Ra+1,R30,Rm	31	A+1	31	M	1	331	1	
					4	RLWINM R30, Ra+1,SH=0,MB=16,ME=31	21	A+1	30	0	16	31	0	
					3	MULLU. Ra,R30,Rm	31	A	30	M	1	235	0	
					2	STW Ra, d(r0)	36	A	0		d			H
					1	STW Ra, d(r0)	36	A	0		d			H

1 LHA Rm,d(r0)
2 LHA R29,d+1(r0); Y=d+1
3 RLWINM R31, Rm,SH=0,MB=28,ME=31
4 BC (true goto 53)
5 COMPI crfD,0,R31, simm=8
6 BC (true goto 20)
7 COMPI crfD,0,R31, simm=A
8 BC (true goto 18)
9 COMPI crfD,0,R31, simm=C
10 BC (true goto 16)
11 COMPI crfD,0,R31, simm=E
12 BC (true goto 14)
13 B goto 50
14 RLWINM R30, Rs2,SH=0,MB=22,ME=24
15 B goto 21
16 RLWINM R30, Rs2,SH=0,MB=20,ME=22
17 B goto 21
18 RLWINM R30, Rs2,SH=0,MB=18,ME=20
19 B goto 21
20 RLWINM R30, Rs2,SH=0,MB=162,ME=18
21 COMPI crfD,0,R30, simm=0 n[16]
22 BC end
23 COMPI crfD,0,R30, simm=10 n[16,17]
24 BC (true goto 27)
25 COMPI crfD,0,R30, simm=11 n[16,17]
26 ADDO R29, R29, Rm
27 LHA R28,d(r29); d=0
28 LHA R26,d(r29); d=1
29 RLWINM R27, R28,SH=0,MB=18,ME=19
30 COMPI crfD,0,R27, simm=0000
31 BC (true goto 42)
32 COMPI crfD,0,R27, simm=1000
33 BC (true goto 41)
34 COMPI crfD,0,R27, simm=2000
35 BC (true goto 39)
36 COMPI crfD,0,R27, simm=3000
37 ADDO R26, R26, Rm+1
38 B goto 42
39 ADDO R26, R26, Rm
40 B goto 42
41 ADDO R26, R26, Rx
42 RLWINM R27, R28,SH=0,MB=17,ME=17
43 COMPI crfD,0,R27, simm=4000
44 BC (false goto 48)
45 LHA R25, d(r26); d=0
46 LHA Ra, d(r25); d=0
47 B goto END

```
48 LHA Ra, d(r26);    d=0
49 B  goto END
50 ADDIC R26, Rm, Y
51 LHA Ra, d(r26);    d=0
52 B  goto END
53 LHA Ra, d(r0);     d=Y
   END

 1 LHA Rm,d(r0)
 2 RLWINM R31, Rm,SH=0,MB=1,ME=6
 3 RLWINM R31, R31,SH=7,MB=25,ME=31
 4 SUBFIC R31, R31, SIMM=64
 5 NEG R31, R31
 6 RLWINM R31, R31,SH=3,MB=22,ME=28
 7 ADDI R31, R31, SIMM=1028
 8 RLWINM R30, Rm,SH=0,MB=0,ME=0
 9 RLWINM R29, Rm,SH=12,MB=0,ME=31
10 CNTLZW R28, R29
11 SUBF R31, R28, R31
12 RLWNM R29, R29, R28, MB=0, ME=31
13 RLWINM R27, R29,SH=20,MB=0,ME=11
14 RLWINM R29, R29,SH=20,MB=12,ME=31
15 OR R31, R30, R31
16 OR R31, R31, R29
17 STW R31, d(r0)
18 STW R27, d+1(r0)
19 LFD frD, d(r0)
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	10 Jun 96	Final 30 Nov 95 - 11 Jun 96

4. TITLE AND SUBTITLE
32-Bit Emulator Chip for High Throughput Processing

5. FUNDING NUMBERS
PR-N00019-96-P6-RE002

6. AUTHOR(S)
Phan, George Q.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Systems & Processes Engineering Corporation (SPEC)
401 Camp Craft Road
Austin, Texas 78746-6558

8. PERFORMING ORGANIZATION REPORT NUMBER
A002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Air Systems Command
AIR-2.5.2
1421 Jefferson Davis Highway
Arlington, VA 22243-5120

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
SBIR Data Rights IAW DFAR 252.227-7018 claimed until June 2001

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A is Superceded by Block 11 until June 2001

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Systems & Processes Engineering Corporation (SPEC) has developed a high throughput emulator chip which provides direct execution of existing AN/AYK-14 standard code on the commercially available PowerPC microprocessor. The SPEC emulator chip allows legacy and PowerPC code to be executed simultaneously, supporting an easy migration path to a full PowerPC code implementation. The SPEC emulator chip has better performance than a traditional software emulation or ROM based design. The SPEC emulator chip has lower fabrication and maintenance cost than using a multiple parallel processor approach. In summary, the SPEC emulator chip has the following features:

- Full AN/AYK-14 emulation
- Simultaneously integrates legacy and new native PowerPC code
- Requires no software modifications
- Direct replacement of the AN/AYK-14 processor card
- Ease in performance validation
- Low cost and maintenance

SPEC can develop drop in replacement circuit cards to meet specific user applications based on its ASIC design implementation. Emulator ASICs can either be implemented in CMOS or GaAs providing up to 500 MHz operation.

14. SUBJECT TERMS
Emulator, AN/AYK-14, PowerPC, ROM

15. NUMBER OF PAGES
92

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL